Handbook of Satisfiability
Armin Biere, Marijn Heule, Hans van Maaren and Toby Walsh
IOS Press, 2021

Chapter 7

Proof Complexity and SAT Solving

Sam Buss and Jakob Nordström

7.1. Introduction

The satisfiability problem (SAT) — i.e., to determine whether a given a formula in propositional logic has a satisfying assignment or not — is of central importance to both the theory of computer science and the practice of automatic theorem proving and proof search. Proof complexity — i.e., the study of the complexity of proofs and the difficulty of searching for proofs — joins the theoretical and practical aspects of satisfiability.

For theoretical computer science, SAT is the canonical NP-complete problem, even for conjunctive normal form (CNF) formulas [Coo71, Lev73]. In fact, SAT is very efficient at expressing problems in NP in that many of the standard NP-complete problems, including the question of whether a (nondeterministic) Turing machine halts within n steps, have very efficient, almost linear time, reductions to the satisfiability of CNF formulas.[1] A popular hypothesis in the computational complexity community is the *Strong Exponential Time Hypothesis (SETH)*, which says that any algorithm for solving CNF SAT must have worst-case running time (roughly) 2^n on instances with n variables [IP01, CIP09]. This hypothesis has been widely studied in recent years, and has served as a basis for proving conditional hardness results for many other problems. In other words, CNF SAT serves as the canonical hard decision problem, and is frequently conjectured to require exponential time to solve.

In contrast, for practical theorem proving, CNF SAT is the core method for encoding and solving problems. On one hand, the expressiveness of CNF formulas means that a large variety of problems can be faithfully and straightforwardly translated into CNF SAT problems. On the other hand, the message that SAT is supposed to be hard to solve does not seem to have reached practitioners of SAT solving; instead, there has been enormous improvements in performance in SAT algorithms over the last decades. Amazingly, state-of-the-art algorithms for deciding satisfiability — so-called *SAT solvers* — can routinely handle real-world

[1] Formally, these reductions run in *quasilinear* time, i.e. time at most $n(\log n)^k$ for some constant k. For these quasilinear time reductions of the Turing machine halting problem to CNF SAT, see [Sch78, PF79, Rob79, Rob91, Coo88].

instances involving hundreds of thousands or even millions of variables. It is a dramatic development that SAT solvers can often run in (close to) linear time!

Thus, theoreticians view SAT as being infeasible, while practitioners view it as being (often) feasible. There is no contradiction here. First, it is possible construct tiny formulas with just a few hundred variables that are totally beyond reach for even the best of today's solvers. Conversely, the large instances which are solved by SAT solvers are based on problems that seem to be "easy" in some sense, although they are very large. However, we currently lack a good theoretical explanation for what makes these problems "easy". Most SAT solvers are general-purpose and written in a very generic way that does not seem to exploit special properties of the underlying problem. Nonetheless, although SAT solvers will sometimes fail miserably, they succeed much more frequently than might be expected. This raises the questions of how practical SAT solvers can perform so well on many large problems and of what distinguishes problems that can be solved by SAT solvers from problems that cannot.

The best current SAT solvers are based on *conflict-driven clause learning (CDCL)* [MS99, MMZ+01].[2] Some solvers also incorporate elements of algebraic reasoning (e.g., Gaussian elimination) and/or geometric reasoning (e.g., linear inequalities), or use algebraic or geometric methods as the foundation rather than CDCL. Another augmentation of CDCL that has attracted much interest is *extended resolution (ER)*. How can we analyze the power of such algorithms? Our best approach is to study the underlying methods of reasoning and what they are able or unable to do in principle. This leads to the study of proof systems such as resolution, extended resolution, Nullstellensatz, polynomial calculus, cutting planes, et cetera. *Proof complexity*, as initiated in modern form by [CR79, Rec75], studies these system mostly from the viewpoint of the complexity of static, completed proofs. With a few exceptions (perhaps most notably *automatability*[3] as defined in [BPR00]), research in proof complexity ignores the constructive, algorithmic aspects of SAT solvers. Nonetheless, proof complexity has turned out to be a very useful tool for studying practical SAT solvers, in particular, because it is a way to obtain mathematically rigorous bounds on solver performance. Lower bounds on the complexity of proofs in a proof systems show fundamental limitations on what one can hope to achieve using SAT solvers based on the corresponding method of reasoning. Conversely, upper bounds can be viewed as indications of what should be possible to achieve using a method of reasoning, if only proof search using this method could be implemented efficiently enough.

We want to stress, though, that the tight connections to proof complexity come with a price. Since proof complexity studies what reasoning can be achieved by a method in principle, ignoring the algorithmic challenge of actually implementing such reasoning, it becomes very hard to say anything meaningful about satisfiable formulas. In principle, it is very hard to rule out that a solver could simply zoom in on a satisfying assignment right away, and this will of course only take linear time. For unsatisfiable formulas, however, a (complete) solver will have to certify that there is no satisfying assignment, and for many SAT

[2] A similar technique for constraint satisfaction problems (CSPs) was independently developed in [BS97].

[3] Automatability is sometimes also called *automatizability*.

solving methods used in practice this is a computational task that is amenable to mathematical analysis. This article, therefore, will focus almost exclusively on unsatisfiable formulas. This is, admittedly, a severe limitation, but it is dictated by the limitations of our current mathematical knowledge. There has been some work on proof complexity of satisfiable formulas by reducing this to problems about unsatisfiable subformulas (e.g., in [AHI05]), but the literature here is very limited. Having said this, we want to point out that the setting of unsatisfiable formulas is nevertheless very interesting, since in many real-world applications proving unsatisfiability is the objective, and conventional wisdom is that such instances are often the hardest ones.

This chapter is intended as an overview of the connection between SAT solving and proof complexity aimed at readers who wish to become more familiar with either (or both) of these areas. We focus on the proof systems underlying current approaches to SAT solving. Our goal is first to explain how SAT solvers correspond to proof systems and second to review some of the complexity results known for these proof systems. We will discuss *resolution* (corresponding to basic CDCL proof search), *Nullstellensatz* and *polynomial calculus* (corresponding to algebraic approaches such as *Gröbner basis* computations), *cutting planes* (corresponding to *pseudo-Boolean solving*), *extended resolution* (corresponding to the *DRAT* proof logging system used for CDCL solvers with pre-/inprocessing), and will also briefly touch on *Frege systems* and *bounded-depth Frege systems*.

We want to emphasize that we do not have space to cover more than a small part of the research being done in proof complexity. Some useful material for further reading are the survey articles [BP98a, Seg07] and the recent book [Kra19]. Additionally, we would like to mention the authors' own surveys [Bus99, Bus12, Nor13]. The present chapter is adapted from and partially overlaps with the second author's survey [Nor15], but has been thoroughly rewritten and substantially expanded with new material.

7.1.1. Outline of This Survey Chapter

The rest of this chapter is organized as follows. Section 7.2 presents a quick review of preliminaries. We discuss the resolution proof system and describe the connection to CDCL SAT solvers in Section 7.3, and then give an overview of some of the proof complexity results known for resolution in Section 7.4. In Section 7.5 we consider the algebraic proof systems Nullstellensatz and polynomial calculus, and also briefly touch on algebraic SAT solving. In Section 7.6 we move on to the geometric proof system cutting planes and the connections to conflict-driven pseudo-Boolean solving, after which we give an overview of what is known in proof complexity about different flavours of the cutting planes proof system in Section 7.7. We review extended resolution and DRAT in Section 7.8, and then continue to Frege and extended Frege proof systems and bounded-depth Frege systems in Sections 7.9 and 7.10. Section 7.11 gives some concluding remarks.

7.2. Preliminaries

We use the notation $[n] = \{1, 2, \ldots, n\}$ for n a positive integer. We write $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$ to denote the set of all natural numbers and $\mathbb{N}^+ = \mathbb{N} \setminus \{0\}$ for the

set of positive integers.

7.2.1. Propositional Logic

A *Boolean variable* x ranges over values *true* and *false*; unless otherwise stated, we identify 1 with true and 0 with false. A *literal* a over a Boolean variable x is either the variable x itself (a *positive literal*) or its negation $\overline{x}$ (a *negative literal*). We define $\overline{\overline{x}} = x$. It will sometimes be convenient to use the alternative notation x^σ, $\sigma \in \{0,1\}$, for literals, where $x^1 = x$ and $x^0 = \overline{x}$. (In other words, x^σ is the literal that evaluates to true under the assignment $x = \sigma$.)

A *clause* $C = a_1 \vee \cdots \vee a_k$ is a disjunction of literals over distinct variables. The empty clause, with $k = 0$, is denoted $\bot$. By convention, clauses are not *tautological*; i.e., do not contain any variable and its negation. A clause C' *subsumes* another clause C if every literal from C' also appears in C. In this case, C' is at least a strong a clause as C. A *k-clause* is a clause that contains at most k literals. A *CNF formula* $F = C_1 \wedge \cdots \wedge C_m$ is a conjunction of clauses. F is a *k-CNF formula* if it consists of k-clauses. We think of clauses and CNF formulas as sets: the order of elements is irrelevant and there are no repetitions.[4]

The *width* of a clause is the number of literals in it. Thus, a *k-clause* has width at most k. In the rest of this chapter, k is assumed to be some arbitrary but fixed constant unless stated otherwise. There is a standard way to turn any CNF formula F into 3-CNF by converting every clause

$$a_1 \vee a_2 \vee \cdots \vee a_w \tag{7.1a}$$

of width $w > 3$ into the set of $w-2$ many 3-clauses

$$\{a_1 \vee a_2 \vee y_2\} \cup \{\overline{y}_{j-1} \vee a_j \vee y_j \mid 3 \leq j < w-1\} \cup \{\overline{y}_{w-2} \vee a_{w-1} \vee a_w\} \ , \tag{7.1b}$$

where the y_j's denote new variables. This conversion to 3-CNF often does not change much from a theoretical point of view (however, there are some exceptions to this rule, which we will discuss in the relevant context).

We typically use N to denote the *size* of a formula F, namely the total number of occurrences of literals in F. (For a k-CNF formula with $k = \mathrm{O}(1)$, N may instead denote the number of clauses in F; this differs from the size of F by at most a constant factor.)

A *truth assignment* is any mapping from variables to truth values 0 or 1. We allow truth assignments to be *partial*, i.e., with some variables left unassigned. A truth assignment is *total* if it assigns values to all variables under consideration. We represent a (partial) truth assignment ρ as the set of literals set to true by ρ. We write $\rho(x^\sigma) = 1$ if $x^\sigma \in \rho$, and $\rho(x^\sigma) = 0$ if $x^{1-\sigma} \in \rho$. If ρ does not assign any truth value to x, we write $\rho(x^\sigma) = *$. The assignment ρ *satisfies* a clause C provided it sets at least one literal of C true; it *falsifies* C provided it sets every literal in C false. It *satisfies* a formula F provided it satisfies every clause in F.

A formula F is *satisfiable* provided some assignment satisfies it; otherwise it is *unsatisfiable*. We say that a formula F *logically implies* a clause C if every total truth assignment ρ which satisfies F also satisfies C. In this case, we write $F \vDash C$. Since $\bot$ is unsatisfiable, $F \vDash \bot$ is equivalent to F being unsatisfiable.

[4] It can be noted, though, that some SAT solvers implement formulas as *multisets* of clauses.

7.2.2. Proof Systems

The principal task of a SAT solver is to determine whether a given formula F is satisfiable or unsatisfiable. When F is satisfiable, there is always a short proof of its satisfiability; namely, a satisfying truth assignment. Establishing that F is unsatisfiable is generally done using some kind of proof system; *proof complexity* studies the complexity of the possible proofs in different proof systems.

In what follows, we give a high-level mathematical description of what a formal proof system is. Later sections will focus in more detail on concrete proof systems that are relevant in the context of SAT solving, and these sections can be read independently without appreciating all of the finer details below.

In the most general setting of proof complexity, one focuses on some *language* L, i.e., an (infinite) set of strings over some (finite) alphabet of symbols, and studies the complexity of proving that a given string x is in the set L. In this abstract setting, a *proof system for* L is a binary predicate $\mathcal{P}(x, \pi)$ that takes as input two strings x and π, is computable (deterministically) in time polynomial in the sizes $|x|$ and $|\pi|$ of the inputs, and has two key properties:

Completeness: for all $x \in L$ there is a string π (a *proof*) for which $\mathcal{P}(x, \pi)$ evaluates to true;

Soundness: for all $x \notin L$ it holds for all strings π that $\mathcal{P}(x, \pi)$ evaluates to false.

Intuitively, completeness means that all valid statements are provable, whereas soundness means that no fake proofs of invalid claims are accepted.

Informally, the strength of a proof system is measured by how small proofs it can produce. The *size* of a formula or proof is equal to the number of symbols it contains. The ideal scenario would be to have a proof system where all theorems can be proven with at most a polynomial blow-up in size of the proof compared to the size of the input. Formally, a proof system is said to be *polynomially bounded* if for every $x \in L$ there is a proof π_x for x that has size at most polynomial in $|x|$.

It is common to measure the relative strengths of proof systems in terms of the lengths of their proofs. Suppose $\mathcal{P}$ and $\mathcal{P}'$ are proof systems. We say that $\mathcal{P}'$ *polynomially simulates* $\mathcal{P}$ if, for all $x \in L$, the size of the shortest $\mathcal{P}'$-proof of x is polynomially bounded by the size of the shortest $\mathcal{P}$-proof of x.[5]

In the context of this survey, we will mostly focus on proving unsatisfiability of CNF formulas. This means that we will deal with proof systems for the formal language L consisting of all unsatisfiable CNF formulas. In this context, a proof system is a polynomial-time algorithm for verifying the unsatisfiability of a given formula F with the aid of a given proof π. Most of the proof systems relevant for SAT solvers (such as resolution, Nullstellensatz, polynomial calculus, cutting planes, et cetera) are such *refutation systems* designed for refuting unsatisfiable formulas F. However, some proof systems in propositional logic (such as Frege systems) instead produce proofs of valid formulas. For such proof system, the language L under study consists of all tautological formulas. There is a duality

[5] In the literature, the terms "simulate", "p-simulate" and "polynomially simulate" are used more or less interchangeably, but sometimes coupled with the proviso that there is a polynomial time algorithm which can produce a $\mathcal{P}'$-proof given a $\mathcal{P}$-proof as input.

between refutations and proofs in that the CNF formula F is unsatisfiable if and only if its negation, the DNF formula $\neg F$, is valid (i.e., is a tautology). We will use the term *proof system* to refer to both kinds of systems above. When discussing refutation systems, we will frequently use the term *proof (of unsatisfiability) for F* to refer to a refutation of F.

7.2.3. Complexity and Asymptotic Notation

One of the central tasks of proof complexity is to prove upper or lower bounds on the complexity of proofs. That is, for a fixed proof system $\mathcal{P}$ and a class $\mathcal{F}$ of unsatisfiable formulas, we may let $f(N)$ denote the worst-case size that might be needed for a $\mathcal{P}$-refutation of any formula $F \in \mathcal{F}$ of size N; then we wish to provide sharp bounds on the growth rate of $f(N)$. Small upper bounds on $f(N)$ indicate that $\mathcal{P}$ is an efficient proof system for $\mathcal{F}$-formulas. Small upper bounds also suggest the possibility of efficient search procedures for finding refutations of formulas $F \in \mathcal{F}$. Large lower bounds on $f(N)$ indicate that $\mathcal{P}$ cannot be efficient for all $\mathcal{F}$-formulas.

Proof complexity often deals with asymptotic upper and lower bounds, generally expressed by using "big-O" notation. The commonly used notations include $f(n) = \mathrm{O}(g(n))$, $f(n) = \mathrm{o}(g(n))$, $f(n) = \Omega(g(n))$, $f(n) = \omega(g(n))$, and $f(n) = \Theta(g(n))$, where $f(n)$ and $g(n)$ are always nonnegative. We write $f(n) = \mathrm{O}(g(n))$ to express that $\exists c{>}0\, \exists n_0\, \forall n{\geq}n_0\, f(n) \leq c \cdot g(n)$, i.e, that f grows at most as quickly as g asymptotically. We write $f(n) = \mathrm{o}(g(n))$ to express that $\forall \epsilon{>}0\, \exists n_0\, \forall n{\geq}n_0\, f(n) \leq \epsilon \cdot g(n)$, i.e., that f grows strictly more slowly than g. When $g(n) > 0$ always holds, $f(n) = o(g(n))$ is equivalent to $\lim_{n\to\infty} f(n)/g(n) = 0$. The notations $f(n) = \Omega(g(n))$ and $f(n) = \omega(g(n))$, mean that $g(n) = \mathrm{O}(f(n))$ and $g(n) = \mathrm{o}(f(n))$, respectively, or, in words, that f grows at least as fast or strictly faster than g. We write $f(n) = \Theta(g(n))$ to denote that both $f(n) = \mathrm{O}(g(n))$ and $g(n) = \mathrm{O}(f(n))$ hold, i.e., that asymptotically speaking f and g are essentially the same function up to a constant factor. Big-O notation is often used in subexpressions. For example, we write $f(n) = 2^{(1-\mathrm{o}(1))n}$ to mean that for any fixed $\epsilon > 0$, we have $f(n) > 2^{(1-\epsilon)n}$ for all sufficiently large n.

7.3. Resolution and CDCL SAT solvers

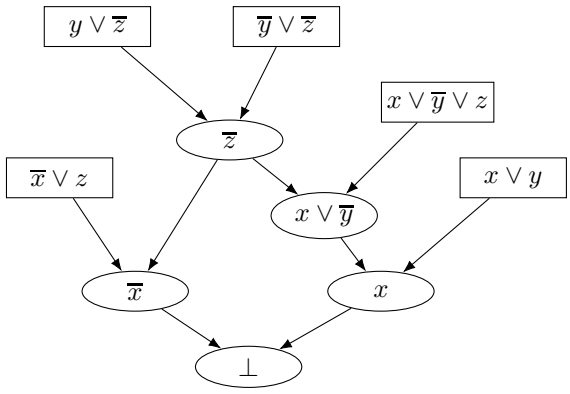
We start our survey by discussing the *resolution proof system* [Bla37, DP60, DLL62, Rob65],which is the most important proof system from the point of view of SAT solving. Resolution is a refutation system that works directly with clauses. A resolution derivation π of a clause D from a CNF formula F is a sequence of clauses $\pi = (D_1, D_2, \ldots, D_{L-1}, D_L)$ such that $D = D_L$ and each clause D_i is either

1. an *axiom clause* $D_i \in F$, or
2. a clause of the form $D_i = B \vee C$ derived from clauses $D_j = B \vee x$ and $D_k = C \vee \overline{x}$ for $j, k < i$ by the *resolution rule*

$$\frac{B \vee x \qquad C \vee \overline{x}}{B \vee C}\ . \tag{7.2}$$

1.	$x \vee y$	Axiom
2.	$x \vee \overline{y} \vee z$	Axiom
3.	$\overline{x} \vee z$	Axiom
4.	$y \vee \overline{z}$	Axiom
5.	$\overline{y} \vee \overline{z}$	Axiom
6.	$\overline{z}$	Res(4, 5)
7.	$\overline{x}$	Res(3, 6)
8.	$x \vee \overline{y}$	Res(2, 6)
9.	x	Res(1, 8)
10.	$\bot$	Res(7, 9)

(a) Refutation as annotated list.



(b) Refutation as directed acyclic graph (DAG).

Figure 7.1: Resolution refutation of the CNF formula (7.3).

We call $B \vee C$ the *resolvent over* x of $B \vee x$ and $C \vee \overline{x}$.

We write $\pi : F \vdash D$ to denote that π is a derivation of D from F; we write $F \vdash D$ to denote that some such π exists. When D is the empty clause $\bot$, we call π a *resolution refutation* of F, and write $\pi : F \vdash \bot$. The *length*, or *size*, of a resolution derivation/refutation is the number of clauses in it.

A resolution derivation π can be represented as a list of clauses annotated with explanations for each clause how it was obtained. This is illustrated in Figure 7.1a for a resolution refutation of the CNF formula

$$(x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (y \vee \overline{z}) \wedge (\overline{y} \vee \overline{z}) \ . \tag{7.3}$$

A derivation $\pi = (D_1, D_2, \ldots, D_L)$ from F can alternatively be represented by a directed acyclic graph (DAG) G_π, also referred to as a *proof DAG*, in the following way. The vertices of G_π are $\{v_1, v_2, \ldots, v_L\}$, with vertex v_i labelled by the clause D_i. The sources (also called *leaves*) of the DAG are vertices labelled with the axiom clauses in F. Without loss of generality there is a unique sink and it is labelled with D_L.[6] If π is a refutation, the sink is labelled with $\bot$. Every vertex that is not a source has indegree two and is the resolvent of its two predecessors. See Figure 7.1b for the refutation in Figure 7.1a represented as a DAG instead.

A resolution refutation π is *tree-like* if G_π is a tree, or equivalently, if every clause D_i in the refutation is used at most once as a hypothesis in an application of the resolution rule. Note that it is permitted that clauses are repeated in the sequence $\pi = (D_1, D_2, \ldots, D_L)$, so different vertices in G_π can be labelled by the same clause if need be (though every repetition counts towards the size of the refutation). The refutation in Figure 7.1b is not tree-like since the clause $\overline{z}$ is used twice, but it could be made tree-like if we added a second derivation of $\overline{z}$ from $y \vee \overline{z}$ and $\overline{y} \vee \overline{z}$. More generally, any derivation can be converted into a

[6] Any other sinks are parts of superfluous subderivations that can be removed.

tree-like derivation by repeating subderivations, but possibly at the cost of an exponential increase in the size of the proof [Tse68, BIW04].

As discussed in Section 7.2.2, *completeness* and *soundness* are fundamental properties of proof systems. For resolution, they are as follows.

Theorem 7.3.1. *Let F be a CNF formula and C be a clause.*

Completeness: *If $F \vDash C$, then there is a clause $C' \subseteq C$ such that $F \vdash C'$. In particular, if F is unsatisfiable, then F has a resolution refutation.*

Soundness: *If $F \vdash C$, then $F \vDash C$. In particular, if F has a resolution refutation, then F is unsatisfiable.*

For technical reasons it is sometimes convenient to allow also the *weakening rule*

$$\frac{B}{B \vee C}\ , \tag{7.4}$$

which makes it possible to infer a subsumed clause (i.e., a strictly weaker clause) from an already derived clause. Resolution with weakening is also sound and complete; in fact, completeness holds in the stronger sense that if $F \vDash C$, then C has a resolution-plus-weakening derivation from F.

It is not hard to show that weakening inferences in a resolution refutation can be eliminated without increasing the complexity of the proof. This also holds for all the notions of complexity that we will discuss later, including length, width, and space.

An important restricted form of resolution is *regular resolution*. A resolution refutation π is *regular* provided that no variable is resolved over more than once along any path in the DAG G_π. For example, the refutation of Figure 7.1 is not regular, since the inferences introducing the clauses $\overline{z}$ and x lie on a common path, and both resolve on y. The soundness of regular resolution is immediate from the soundness of resolution. Regular resolution is also complete [DP60]. However, it was shown in [Urq11, AJPU07] that there is a family of CNF formulas F_n that have polynomial size resolution refutations, but are such that, letting $c = c(n)$ be the length of F_n (i.e., the number of clauses in F_n), the shortest regular resolution refutation of F_n has size $2^{c^{1-o(1)}}$. This gives an exponential separation between regular resolution and (general) resolution. In particular, regular resolution does not efficiently simulate resolution in the sense discussed in Section 7.2.2.

As already mentioned, resolution without weakening can polynomially simulate resolution. On the other hand, tree-like resolution cannot polynomially simulate resolution, or even regular resolution [Tse68].

Another interesting restriction on resolution which is relevant for CDCL SAT solvers is *trivial resolution*. A resolution derivation π is a trivial derivation of D from F if it can be written as a sequence of clauses $\pi = (D_1, \ldots, D_L = D)$ such that

1. π is *input*; i.e., $D_1 \in F$, and for all $i \geq 1$ it holds that $D_{2i} \in F$ and that D_{2i+1} is the resolvent of the latest derived clause D_{2i-1} with the axiom D_{2i}.
2. π is regular, i.e., no variable is used more than once as the resolution variable.

The sequence π is a trivial resolution refutation provided D_L is $\bot$. Trivial resolution is sound but *not* complete. For example (and jumping ahead a bit), the formula (7.3) is not refutable by *unit propagation* (which will be described later), and thus does not have a trivial resolution refutation.

A *unit clause* is a clause of width 1. A *unit resolution* refutation is a resolution refutation in which each resolution inference has at least one unit clause as a hypothesis. Unit resolution is sound but not complete. Indeed, it is very closely connected to trivial resolution [BKS04]. For F a CNF formula and $C = a_1 \vee \cdots \vee a_k$ a clause, there is a trivial resolution derivation of C from F if and only if there is a unit resolution refutation of $F \cup \{\overline{a}_1, \overline{a}_2, \cdots, \overline{a}_k\}$, i.e., of F plus k many unit clauses. In particular, F has a trivial resolution refutation exactly when F has a unit resolution refutation.

7.3.1. DPLL and Conflict-Driven Clause Learning

The most successful present-day SAT solvers are based on *conflict-driven clause learning (CDCL)* [MS99, MMZ+01], which, when run on unsatisfiable CNF formulas, is a search procedure for resolution proofs. CDCL algorithms have been highly refined and augmented with both practical optimizations and sophisticated inference techniques; however, they are based on four main conceptual ingredients:

1. *DPLL*, a depth-first, backtracking search procedure for satisfying assignments.[7]
2. *Unit propagation*, a method for making obvious inferences during search.
3. A *clause learning* algorithm that attempts to prune the search space by inferring ("learning") clauses whenever the search procedure falsifies a clause (finds a "conflict") and has to backtrack.
4. A *restart policy* for stopping a depth-first search and starting a new one.

We will discuss these four ingredients one at a time. Our treatment will be informal, without including all necessary details, as we presume that most readers are familiar with the concepts, and as our focus is on discussing connections between SAT solving and proof complexity. For a more detailed treatment, see Chapter 7.11 on complete algorithms and Chapter 7.11 on CDCL solvers.

7.3.1.1. Depth-First Search Using the DPLL Method

The first ingredient, *DPLL*, is named after the four collective authors, Davis, Putnam, Logemann and Loveland, of two of the primary sources for resolution and automated solvers, namely [DP60] and [DLL62]. The DPLL procedure is given as input a CNF formula F, i.e., a set of clauses. Its task is to determine whether F is satisfiable, and if so to find a satisfying assignment. The DPLL algorithm maintains a partial truth assignment ρ and runs the algorithm shown in Figure 7.2 recursively. Initially, ρ is the empty assignment with all variables

[7] Often unit propagation is included as part of DPLL; however, we adopt the convention that DPLL means a backtracking search, and does not need to include unit propagation. The original definition of the DPLL algorithm in [DLL62], following [DP60], included both unit propagation and the pure literal rule.

```
if the partial assignment ρ falsifies some clause of F then
  return false;
end
if ρ satisfies F then
  Output ρ as a satisfying assignment and terminate.
end
Pick some unassigned literal, a, called the decision literal;
Extend ρ to set a true;
Call this DPLL procedure recursively;
Update ρ to set a false;
Call this DPLL procedure recursively (again);
Update ρ to make a unassigned;
return false;
```

Figure 7.2: DPLL recursive procedure (without unit propagation).

unassigned. Each time the recursive procedure is called it chooses a *decision literal* a, and tries setting it first true and then false, using recursive calls to explore the two assignments to a. It either halts with a satisfying assignment, or eventually returns *false* from the top level if F is unsatisfiable.

There is an exact correspondence between the DPLL search procedure (without unit propagation) and tree-like, regular resolution refutations.[8] In one direction, any tree-like regular resolution refutation π containing S occurrences of clauses corresponds to a DPLL search tree with S nodes (each node an invocation of the recursive procedure). Such a search tree can be generated by letting the DPLL search procedure traverse the proof tree G_π starting at $\bot$ in a depth-first fashion. When the traversal of G_π is at a node labelled with a clause C, the partial assignment ρ falsifies C. Conversely, any DPLL search with S calls to the recursive procedure gives rise to a tree-like, regular resolution refutation with $\leq S$ clauses. Such a refutation can be formed from the leaves starting with clauses falsified by the deepest recursive calls. Pairs of recursive calls are joined by resolution inferences as needed (possibly with parts of the DPLL search tree being pruned away).

7.3.1.2. Unit Propagation

The second ingredient is *unit propagation*. Unit propagation can be viewed as a way to guide the DPLL search procedure, but it becomes much more important when used with clause learning, as will be discussed momentarily. Suppose that the CNF formula F contains a clause $C = a_1 \vee \cdots \vee a_k$ and that the current partial assignment ρ has set all but one of the literals in C false, and that the remaining literal a_i is unassigned. Then *unit propagation* is the operation of setting a_i true and adding it to ρ. This can be done without loss of generality, since setting a_i false would falsify C.

Unit propagation is directly related to the unit resolution proof system discussed earlier. Let F' be the CNF formula consisting of F plus the unit clauses $\overline{a}$ for all literals a such that $\rho(a) = 0$. Then repeatedly applying unit propagation

[8] Any tree-like resolution refutation can be converted to be regular by pruning away parts of the proof where variables are resolved on more than once.

```
ρ0 ← ρ;
Extend ρ by unit propagation for as long as possible;
if ρ falsifies some clause of F then
    ρ ← ρ0;
    return false;
end
if ρ satisfies F then
    Output ρ as a satisfying assignment and terminate.
end
Pick some literal a not set by ρ (the decision literal);
Extend ρ to set a true;
Call this DPLL procedure recursively;
Update ρ to set a false;
Call this DPLL procedure recursively (again);
ρ ← ρ0;
return false;
```

Figure 7.3: The recursive procedure for DPLL with unit propagation.

starting with F and the partial assignment ρ will falsify some clause of F if and only if F' has a unit resolution refutation.

With unit propagation, the recursive procedure for the DPLL algorithm is shown in Figure 7.3. This runs similar to the algorithm of Figure 7.2, but now does unit propagation whenever possible. Values of ρ can be set either as decision literals or by unit propagation (with ρ_0 used to restore the state of ρ when returning). Each set literal is assigned a *decision level*. Literals propagated even before any decision literal has been set are assigned decision level 0. The decision level is incremented when setting a decision literal, and unit propagated literals are given the decision level of the last decision literal. Each call to the procedure either halts with a satisfying assignment extending ρ, or returns false if no such satisfying assignment exists.

7.3.1.3. Clause Learning and Nonchronological Backtracking

The third ingredient in the recipe for state-of-the-art SAT solvers is a crucial one: the use of *clause learning* to infer new clauses. This modifies the first step of DPLL algorithm to also add clauses to the formula; namely, when ρ falsifies some clause of F, i.e., reaches a *conflict*, then clause learning is used to derive one or more clauses and add them to F. The intent is that the learned clauses will be used in the future to prune the search space and avoid repeatedly exploring the same impossible assignments to literals. Essentially all clause learning methods used in practice will infer only clauses C that can be derived from F (together with previously learned clauses) by trivial resolution. This is due to the equivalence between trivial resolution refutations and unit resolution refutations and the fact that DPLL/CDCL uses unit propagation to find contradictions.

The most popular methods for learning clauses in modern CDCL SAT solvers are based on the *first unique implication point (1UIP)* learning scheme, as illustrated in Figure 7.4. But for purposes of the connection between clause learning and trivial resolution, the key point is that the learned clause is chosen to be a

clause $C = a_1 \vee \cdots \vee a_k$ with the following two properties:

1. C is falsified by ρ, i.e., $\rho(a_i) = 0$ for each a_i in C, and
2. there is a unit resolution refutation of F plus the k unit clauses $\{\overline{a}_i \mid i \in [k]\}$.

When C is chosen in this way, C is called a *reverse unit propagation (RUP)* clause or an *asymmetric tautology (AT)* for F.[9] By the correspondence between trivial resolution and unit resolution, this is equivalent to the existence of a trivial resolution derivation of C from F.

A third property that typically holds for a learned clause C is the *UIP* or *assertive property*:

3. C contains exactly one literal a_i set to false at the last decision level; the rest of the literals in C were set false at earlier decision levels. The literal a_i is called the *UIP literal* or *asserting literal*.

When C is chosen so that a_i is as "close as possible to the contradiction," then this is called 1UIP learning, and C is called the *first unique implication point (1UIP) clause* with a_i being the *1UIP literal*. There is always at least one UIP clause, since the decision literal at the last level is a UIP literal (because this decision led to a conflict).

In order to visualize conflict analysis, it is helpful to construct a *conflict graph* that shows how assignments to different literals causes propagations of other literals. Figure 7.4 illustrates 1UIP clause learning in the conflict graph formed from the clauses

$$\{\overline{x} \vee \overline{a} \vee z,\ \overline{x} \vee \overline{z} \vee y,\ \overline{y} \vee t,\ \overline{y} \vee \overline{a} \vee u,\ \overline{y} \vee \overline{u} \vee v,\\ \overline{y} \vee \overline{t} \vee s,\ \overline{u} \vee \overline{b} \vee \overline{c} \vee w,\ \overline{t} \vee \overline{v} \vee \overline{w},\ \overline{a} \vee \overline{b} \vee c\}\ . \tag{7.5}$$

Arrows indicate which literals are involved in causing which unit propagations. The three literals a, b, c have been set true at earlier decision levels, and the decision literal at the last level is x. The 1UIP learned clause is $\overline{a} \vee \overline{b} \vee \overline{c} \vee \overline{y}$, with the 1UIP literal being y. The dashed line shows the portion of the conflict graph with propagations that depend on y, and the predecessors vertices of all edges crossing this cut are the literals a, b, c, and y that form the learned clause.

For an alternative view of the same conflict analysis, Figure 7.5a shows the unit propagations caused by setting x to 1, where $x \xleftarrow{\mathsf{DEC}} 1$ denotes that x is set true as a decision literal and $\ell \xleftarrow{C} \sigma$ denotes the literal ℓ being set to the truth value σ by unit propagation using clause C. The 1UIP clause $\overline{a} \vee \overline{b} \vee \overline{c} \vee \overline{y}$ is derived as depicted in Figure 7.5b by traversing backwards through the list of unit propagated literals starting with the clause $\overline{t} \vee \overline{v} \vee \overline{w}$ that was falsified. Each time a literal is reached that is set false in the current clause, a resolution inference is applied to update the current clause. (Note how s is skipped as it does not appear negated in $\overline{b} \vee \overline{c} \vee \overline{t} \vee \overline{u} \vee \overline{y}$.) When we reach for the first time a clause (in this example, $\overline{a} \vee \overline{b} \vee \overline{c} \vee \overline{y}$) that contains only a single literal at the last decision level (in this example, y), we have obtained the 1UIP clause.

A nice feature of the 1UIP clause learning scheme, and the reason that the learned clause is called *asserting*, is that this clause contains exactly one literal a_i

[9] See [GN03, Van08] for the original definitions of RUP; the modern terminology AT is discussed in [HJB10, JHB12].

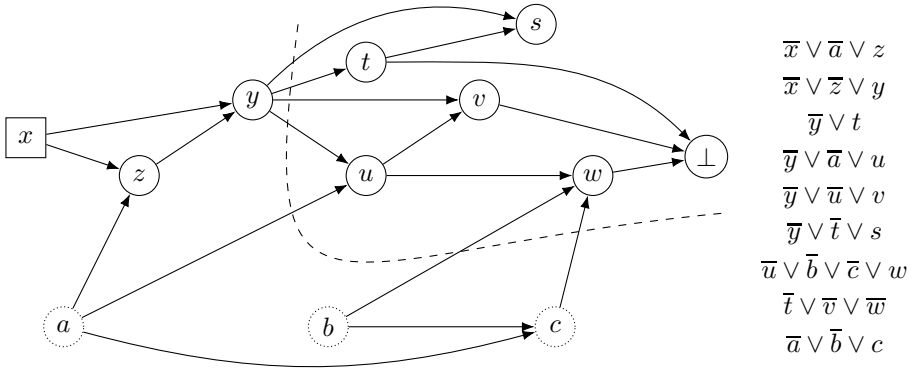


Figure 7.4: Example of a conflict graph and 1UIP learning with learned clause $\overline{a} \vee \overline{b} \vee \overline{c} \vee \overline{y}$. This conflict graph is generated using the clauses in Equation (7.5), also shown on the right in the figure.

that has not been set false at an earlier decision level. This means that after the solver has backtracked and undone the last decision, the literal a_i will unit propagate to true, flipping the value it had before. For example, the clause learned in Figure 7.4 is asserting: a, b and c have been set at earlier levels, and so $\overline{y}$ can be inferred from the learned clause by unit propagation. Conflict analysis algorithms with this property are sometimes referred to as *asserting clause learning schemes*, and learning schemes used in modern CDCL solvers tend to be of this type.

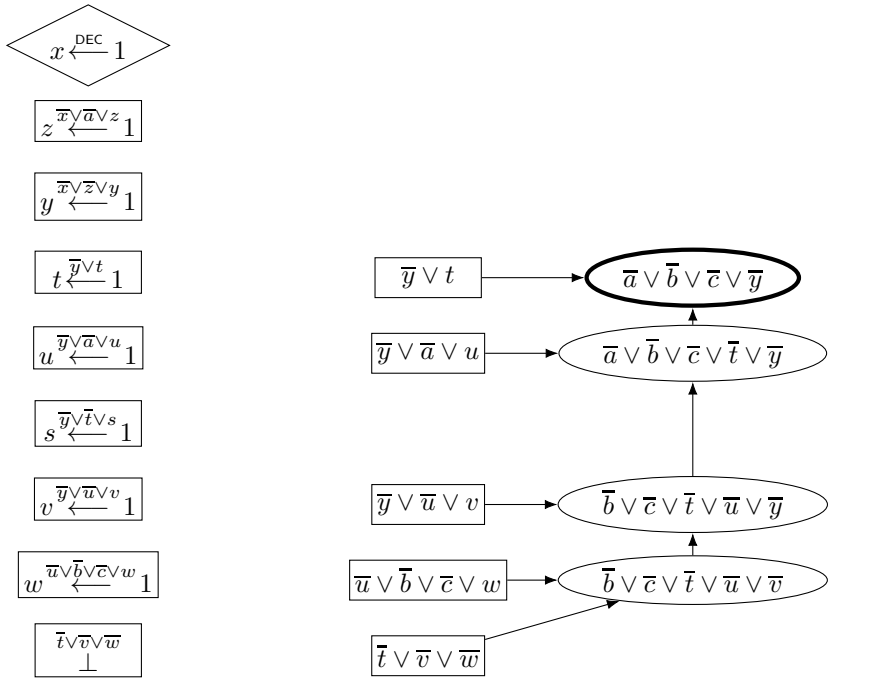
Perhaps the most direct way to incorporate both unit propagation and clause learning (but not backjumping or restarts) in DPLL is as in the algorithm shown in Figure 7.6, which we will call *DPLL with clause learning (DPLL-CL)*. We want to emphasize that DPLL-CL is not an algorithm that would be used in practice; we instead introduce it as a didactic tool to illustrate how there is a natural algorithmic development from DPLL to CDCL.

The DPLL-CL algorithm has been formulated to be faithful to the idea of DPLL as a depth-first search procedure, but has been updated with clause learning. Like the basic DPLL algorithm, it acts to set a literal a first true and then false. The first recursive call to DPLL-CL has the decision literal a set true. If that call returns, the newly learned clauses may already be enough to set a false by unit propagation. But if not, a can be set false as a decision literal anyway. For example, in the 1UIP learning in Figures 7.4 and 7.5 the learned clause allows y to be set false by unit propagation, but it does not allow $\overline{x}$ to be derived by unit propagation. Nonetheless $\overline{x}$ is a consequence of the literals set in ρ.[10]

The DPLL-CL algorithm in Figure 7.6 still lacks an important component of the CDCL algorithm, namely *backjumping* (also called *nonchronological backtracking*). What this means is that the solver backtracks not just one decision level but potentially multiple decision levels at once.[11] For an example of a com-

[10] This will not generally be true in the CDCL algorithm described next, since CDCL allows backjumping to backtrack multiple decision levels.

[11] It can be noted, though, that a CDCL algorithm with chronological backtracking, somewhat



(a) The unit propagations triggered by setting x true.

(b) The trivial resolution derivation of $\overline{a} \vee \overline{b} \vee \overline{c} \vee \overline{y}$ (written bottom-up).

Figure 7.5: Unit propagation derivation of the contradiction shown in Figure 7.4 and trivial resolution derivation of the learned clause (from the clauses in (7.5)).

mon way to use backjumping, refer to the conflict graph and 1UIP learned clause of Figures 7.4 and 7.5. There, x has been set as a decision literal with decision level $lev(x)$. The literals a, b and c were set with strictly lower decision levels $lev(a)$, $lev(b)$ and $lev(c)$; let $L \leq lev(x) - 1$ denote the maximum of these three levels. Once the 1UIP clause $\overline{a} \vee \overline{b} \vee \overline{c} \vee \overline{y}$ has been learned, the literal y can be propagated to true already at decision level L, which is called the *assertion level*. To readily take advantage of this, the CDCL algorithm uses backjumping (nonchronological backtracking) to backtrack to decision level L, which could be much smaller than $lev(x) - 1$. Another reason for backjumping is that it can be viewed a miniature version of restarting, which we will discuss next, in that a restart consists of backtracking to decision level 0.

7.3.1.4. Restarts

The fourth ingredient in our CDCL list is the use of *restarts*. A restart consists of halting the CDCL search, while preserving the clauses learned from F so far, then starting a new CDCL search . As just noted, this already fits into the CDCL

similar in spirit to the DPLL-CL algorithm in Figure 7.6, was recently proposed in [NR18] and has been shown to perform very well on certain types of benchmarks.

```
ρ0 ← ρ;
loop
    Extend ρ by unit propagation for as long as possible;
    if ρ falsifies some clause of F then
        Optionally learn one or more clauses C and add them to F;
        ρ ← ρ0;
        return false;
    end
    if ρ satisfies F then
        Output ρ as a satisfying assignment and terminate.
    end
    Pick some literal a not set by ρ (the decision literal);
    Extend ρ to set a true;
    Call DPLL-CL recursively;
    Unset the value of a in ρ;
    Extend ρ by unit propagation for as long as possible;
    if the value of a is set in ρ (if so, it is false) then
        continue (with the next loop iteration);
    end
    Update ρ to set a false;
    Call DPLL-CL recursively (again);
    ρ ← ρ0;
    return false;
end loop
```

Figure 7.6: The recursive procedure DPLL-CL for DPLL with clause learning.

algorithm described earlier, as setting the new (backjumping) decision level L' equal to 0 causes a restart. The new CDCL search can use different decision literals, and this may provide an advantage in searching for either a satisfying assignment or a refutation. In practice, the use of restarts is crucial for CDCL solver performance. As is discussed later in this section, there are also theoretical reasons why restarts may be beneficial. There are several intuitive explanations of why restarts might be useful, but ultimately, it comes down to the fact that doing a restart allows the CDCL search to try out a different choice of decision literals. There have been extensive investigations of how to do *adaptive restarts* [AS12, Bie08] and how to choose decision literals, e.g., by using *variable state independent decaying sum (VSIDS)* [MMZ$^+$01], *variable move to front (VMTF)* [Rya04], and *phase saving* [PD07]. Going into details about this is far beyond the scope of this survey, but see, e.g., [BF15] for an evaluation of different decision strategies, and [AS12, BF19, Hua07] for good discussions of restart strategies.

7.3.1.5. Putting It All Together

We are now ready to present pseudo-code for an abstract formulation of the CDCL algorithm. Unlike our earlier algorithms, this one is not implemented as a recursive procedure; this is because backjumping allows backtracking multiple decision levels at once. Instead, the CDCL algorithm maintains a current decision level L. Whenever a literal b is set, either as a decision literal or by unit propagation, it is given the current decision level as its level, denoted $lev(b)$. The CDCL algorithm is shown in Figure 7.7; its input is a formula F.

```
L ← 0;
ρ ← empty assignment;
loop
    Extend ρ by unit propagation for as long as possible;
    if ρ satisfies F then
        return ρ as a satisfying assignment;
    else if ρ falsifies some clause of F then
        if L = 0 then
            return "Unsatisfiable";
        end
        Learn one or more clauses C and add them to F;
        Let L′ < L be the minimal assertion level among the clauses learned;
        Unassign all literals set at levels > L′;
        L ← L′;
    else
        Pick some unassigned literal a (the decision literal);
        L ← L + 1;
        Extend ρ to set a true;
    end
    continue (with the next iteration of the loop);
end loop
```

Figure 7.7: The CDCL algorithm (in skeletal form and without clause deletions).

Our pseudo-code does not specify how to implement clause learning. A typical implementation will learn a 1UIP clause (with some postprocessing that we do not discuss here); this clause is asserting in that it allows a new literal to be set by unit propagation. The backjump level L' is then chosen as the assertion level of this clause, i.e., the minimum decision level at which this literal can be unit propagated. The partial assignment ρ will set a literal a true at level 0 if and only if the unit clause a is in F. It follows that if the CDCL algorithm returns "Unsatisfiable," then the current set of clauses F is unsatisfiable. Since clause learning only learns clauses that are consequences of earlier clauses, this means that also the original (input) formula F is unsatisfiable.

If the CDCL algorithm uses 1UIP clause learning, or other clause learning schemes based on the conflict graph, then the learned clauses will be derivable by trivial resolution derivations (by the correspondence between trivial resolution and unit resolution). Thus, if the CDCL algorithm returns "Unsatisfiable", the original formula F has a resolution refutation of size polynomial in the number of clauses learned by the CDCL algorithm.

Figure 7.8b shows an example of a full CDCL refutation for the formula

$$(\overline{u} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee z) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{u} \vee w) \wedge (\overline{u} \vee \overline{w}) \quad (7.6)$$

based on 1UIP clause learning. Decision variables are shown in diamonds. Unit propagations are shown in rectangles. Dashed diamonds and boxes indicate assignments that were preserved after the previous conflict. Darker ovals indicate learned clauses. As shown, the CDCL search finds three conflicts. The first conflict is found after $\overline{w}$ and $\overline{x}$ are chosen as decision literals at decision levels 1 and 2. The resulting learned clause $u \vee x$ is asserting, so x is set true by unit propagation at level 1. The second conflict learns the unit clause $\overline{x}$, which is an asserting

clause setting x false at level 0. The CDCL solver then performs backjumping (nonchronological backtracking); namely, it backtracks to unset all values set at decision levels above the level of the asserted literal x. In this case, backjumping involves backtracking out of decision level 1, unassigning w and u, so that w is no longer a decision literal. The third conflict arises at level 0, and completes the CDCL procedure. However, for pedagogical reasons — to get a clearer connection to the resolution proof — the illustration assumes that the solver does not terminate immediately after detecting a conflict at decision level 0, but instead performs a final round of conflict analysis to formally derive the empty clause $\bot$. Figure 7.8c shows the corresponding resolution refutation of F. Note that each conflict forms a trivial resolution derivation relative to the current CNF formula F as augmented with learned clauses.

7.3.1.6. Clause Deletions

It should be mentioned that one aspect of CDCL solving that is absolutely crucial in practice, but that our discussion above completely ignores, is the use of *clause deletion*, or *clause erasure*. Clause deletion means removing some of the learned clauses. This helps reduce memory usage and, even more importantly, allows unit propagation to run faster. The disadvantage is that clause erasure might delete clauses that would have been useful in the future (but results in [KN20] indicate that in some cases erasing clauses can actually help the solver find shorter resolution refutations). Some theoretical results related to clause deletion will be discussed later in the context of resolution space in Sections 7.4.3 and 7.4.4.

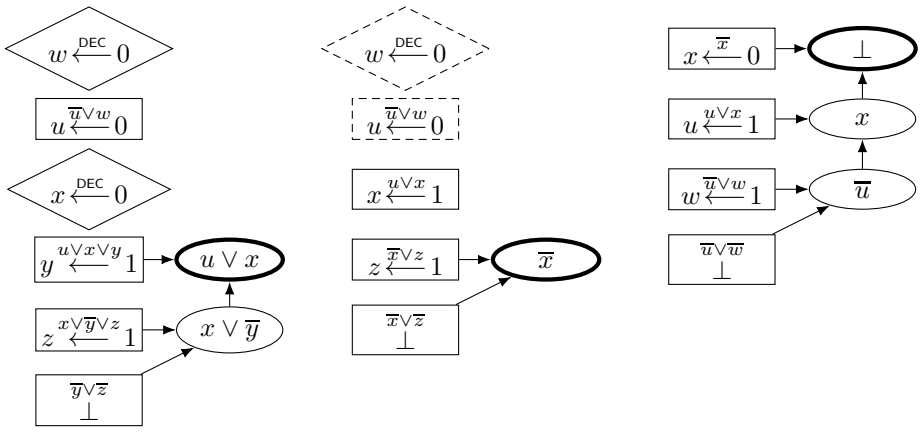
7.3.2. Proof Logging in CDCL Solvers

The output of a SAT solver given a CNF formula F will be either a satisfying assignment for F or the assertion that F is unsatisfiable. In the second case, it can be useful to not just assert that F is unsatisfiable, but also to produce a refutation of F so that this assertion can be verified. Of course, the execution of the SAT solver can serve as a kind of proof, but this is not ideal; first because there could be bugs in the design or implementation of the SAT solver, and second because it does not correspond to a useful proof system. A better option is to output a resolution refutation: this is always possible for the CDCL constructions we have discussed so far, as illustrated in Figure 7.8.

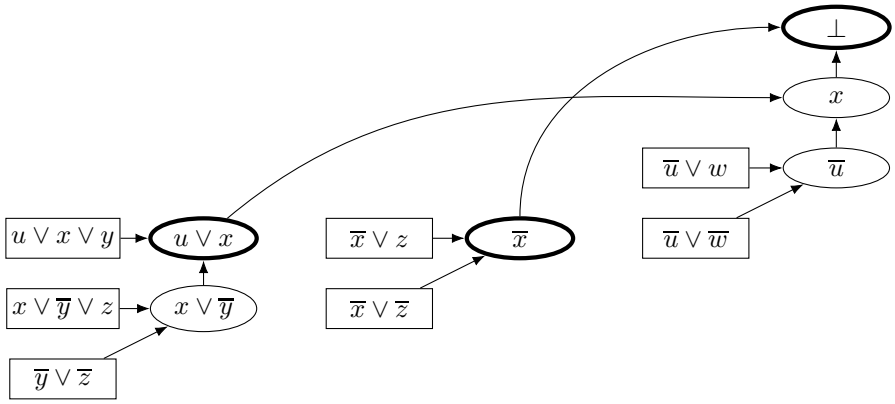
However, the fact that CDCL solvers (implicitly) produce resolution proofs does not necessarily mean that this is the best way of producing certificates of correctness. For a large formula the resolution proof can get very long, containing lots of details that are not really necessary for efficient verification. Also, forcing the solver to output all steps in every conflict analysis can incur a large overhead in running time. A way to get more concise certificates (see [GN03, Bie06, Van08, HHW13a]) is to use a *proof trace* consisting only of the sequence of learned clauses generated during the CDCL search (preferably with unneeded learned clauses omitted). For instance, in the example in Figure 7.8, the proof trace is just the sequence of clauses $u \vee x$, $\overline{x}$, $\bot$. Such a proof is called a *RUP proof*, since each clause follows by reverse unit propagation from F plus the preceding clauses in the sequence. The property of being RUP clause is checkable in polynomial time

$$(\overline{u} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee z) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{u} \vee w) \wedge (\overline{u} \vee \overline{w})$$

(a) The unsatisfiable CNF formula in (7.6).



(b) A CDCL refutation of the CNF formula in Figure 7.8a.



(c) The corresponding resolution refutation.

Figure 7.8: Part (a) shows the unsatisfiable CNF formula (7.6). Part (b) shows a complete CDCL run establishing the unsatisfiability of this CNF formula. Part (c) shows the resolution refutation corresponding to this CDCL execution.

using unit propagation; thus the correctness of a RUP proof trace can be checked in time polynomially bounded by the size of F and the total size of the proof trace.

A very important aspect in applied SAT solving, which is also relevant for proof logging, is that extensive *preprocessing* of the input is performed before the main CDCL search algorithm starts, and some solvers, such as *Lingeling* [Lin], *CryptoMiniSat* [Cry], and *CaDiCaL* [CaD], even interleave preprocessing techniques with CDCL search; this is known as *inprocessing* [JHB12]. This involves using a number of techniques which, although known to be theoretically very bad in the worst case, are indispensable in practice. Many of these techniques can be formalized within resolution and hence in RUP proofs, but not all. Nowadays, RUP proof traces have therefore been supplanted by more sophisticated DRAT proofs [HHW13b]. DRAT proofs generalize RUP proofs, but can also simulate stronger methods of reasoning — in fact, all the way up to extended resolution,[12] as will be discussed later in Section 7.8.

7.3.3. Efficiency of CDCL Proof Search

Now that we have given formal descriptions of different methods for SAT solving, we can study the complexity of proofs as generated by these methods when they are used on unsatisfiable CNF formulas. As discussed above, the DPLL method corresponds to tree-like resolution, which can be exponentially worse than (general, DAG-like) resolution. Since CDCL is only looking for structurally very restricted proofs, it is natural to ask how efficient CDCL proof search can be compared to the best possible general resolution proof. (Henceforth, when we talk about CDCL, we mean CDCL *with restarts*, unless explicitly stated otherwise.) Note that if the formula is exponentially hard for resolution, then basic CDCL search cannot be expected to run fast since it is searching for a resolution proof. But what we can do is to benchmark the solver against the best possible outcome, i.e., the most efficient proof. Thus, we can ask whether a CDCL search strategy can implement efficient proof search in the sense that the algorithm finds a refutation which is no more than polynomially longer than, say, the shortest possible resolution refutation.

This turns out to be a deep and difficult question to answer. What is known is that there is no algorithm that can find resolution proofs efficiently unless $\mathsf{P} = \mathsf{NP}$ [AM19] (this recent paper strengthens previous results in [AR08, MPW19]). In formal proof complexity terminology, resolution is said to be *non-automatable*. What we can do instead is to seek bounds on the proof-theoretic strength of so-called *nondeterministic*[13] solvers that are assumed to somehow magically make good choices for certain heuristics during the search. In a line of works including

[12] It is important to note, however, that the power of DRAT proofs to simulate extended resolution is (at least as of yet) a purely theoretical result. Many reasoning techniques that, while powerful, fall well short of extended resolution, cannot currently be formalized in a practically efficient way in DRAT, two notable examples being cardinality reasoning and Gaussian reasoning. While it is possible in principle to handle cardinality reasoning or Gaussian reasoning in the DRAT proof system [HB18, BT19, HKB20], the attempts made so far to implement this in practice in state-of-the-art CDCL solvers have been far too inefficient.

[13] Note that the word "nondeterministic" is used here in the technical sense of the definition of the complexity class NP, where a computation can split into two branches at every step and

[BKS04, HBPV08, BHJ08] and culminating in the papers [PD11, AFT11], it was shown that nondeterministic CDCL can be as efficient as the resolution proof system except possibly for a polynomial blow-up.

More technically speaking, the assumptions needed are that the CDCL solver is allowed to magically choose decision variables and values to assign to these variables, and also keeps every single clause ever learned during the search. Furthermore, it has to make somewhat frequent restarts, but not more frequent than what is already standard in state-of-the-art solvers, and has to use some asserting clause learning scheme (in particular, 1UIP works fine, but a strength of the simulation is that any other asserting scheme will also be good enough). With these assumptions, the main result in [PD11] is that a CDCL solver can establish unsatisfiability of a CNF formula in time polynomial in the shortest resolution refutation of that formula. One possible way of interpreting this result is that in cases where the decision heuristic works well enough, and when the right learned clauses are kept in memory, then a CDCL solver could in principle run fast for formulas that possess short resolution proofs (although it follows from [AM19] that for any concrete choice of heuristics there are tricky formulas that will make these heuristics fail unless $\mathsf{P} = \mathsf{NP}$).

The construction of [PD11] showing that CDCL simulates resolution can be summarized as follows. Suppose that the formula F has a resolution refutation $\pi = (D_1, \ldots, D_L = \bot)$. Intuitively, the goal for the CDCL search is to successively learn the clauses D_i for $i = 1, 2, \ldots$ until it reaches $D_L = \bot$, which corresponds to a conflict at decision level 0 (as in the example in Figure 7.8). Learning exactly the clauses D_i might not always be possible, but a key insight in [PD11] is that the CDCL solver can instead learn other clauses which together are as powerful as D_i. For this, using a definition from [AFT11], we say that a set of clauses F' *absorbs* a clause $D = a_1 \vee \cdots \vee a_k$ provided that setting any $k-1$ literals of D false allows either the remaining literal of D or the empty clause to be derived by unit propagation from the clauses of F'. As a small, concrete, example, the set of clauses $\{x_1 \vee y_1, \overline{y}_1 \vee x_2 \vee y_2, \overline{y}_2 \vee x_3\}$ absorbs $x_1 \vee x_2 \vee x_3$, since falsifying any two of the x_i-variables will propagate the third to true for both the clause and the clause set.

What absorption means is that it seems to an outside observer that the set of clauses F' contains D, since all unit propagations that would occur if D were in F' can be observed now as well. The construction of [PD11] allowing CDCL search to simulate the resolution refutation π is based on showing that the clause learning can work well enough to to successively absorb the clauses $D_1, \ldots, D_L$ in the refutation π. Suppose that the solver has added enough learned clauses to the clause database $F' \supseteq F$ to have absorbed $D_1, \ldots, D_{i-1}$. To absorb D_i, the CDCL solver repeatedly tries to falsify all literals in D_i. When this is no longer possible to do without the last unassigned literal in D_i being propagated, the clause has been absorbed. As long as it is possible to falsify D_i, then since the clauses resolved to obtain D_i are already absorbed by the inductive hypothesis, it follows that unit propagation will lead to a conflict. At this point, the solver can

succeeds in finding a solution if one of the (potentially exponentially many) branches does so. It thus has a very different meaning from "randomized," where the requirement is that the successful branch should be found with high probability.

add a new learned clause to the clause database F' and restart. It can be shown that this process of falsifying D_i and adding a new learned clause can repeat at most a polynomial number of steps before D_i is absorbed.

The independent work [AFT11] obtained an alternative, more effective version of the simulation result by showing that if a formula F has a resolution refutation in bounded width (i.e., where every clause contains only a small number of literals), then CDCL using a decision strategy with enough randomness will decide F efficiently. At first sight this might not seem so impressive — after all, it is easy to see that in this case exhaustive search in bounded width also runs fast — but the point is that a CDCL solver is very far from doing exhaustive width search and does not care at all about the existence or non-existence of resolution refutations with only small clauses.

A downside of both of these results is that it is crucial for the SAT solver never to delete clauses. This is a very unrealistic assumption, since modern solvers typically throw away a majority of the clauses learned during search. It would be nice to extend the model of CDCL in [AFT11, PD11] to capture memory usage in a more realistic way, and then study the question of whether CDCL can simulate resolution efficiently with respect to both time and space. An attempt in this direction was made in [EJL$^+$16], but the following problem still remains open. (We refer to Section 7.4.3 for the formal definitions of space.)

Open Problem 7.1. Suppose that F is a CNF formula that has a resolution refutation in simultaneous length L and space s. Does this imply that there is a CDCL proof of the unsatisfiability of F (as modelled in [AFT11, PD11, EJL$^+$16]) in simultaneous length and space polynomial in L and s, respectively?

Another downside of [AFT11, PD11] is that the simulation of resolution by CDCL depends crucially on the choice of decision literals. However, the correct choice of decisions is made in a highly non-constructive way — namely, as discussed above, it consists of falsifying the literals in the clauses in the (unknown) shortest resolution refutation one at a time in the order they appear in the refutation, and to do this repeatedly for each clause until it is guaranteed to be absorbed. In other words, the usual heuristics for decisions (such as VSIDS and phase saving) do not come into play at all for the simulation of resolution by CDCL. Similarly, the concrete restart heuristic used plays no role — all that matters is that restarts are frequent enough. The way the construction goes, the solver makes progress towards its goal at the first conflict after every restart, but then it might not do anything useful until the next restart happens. One can show, though, that the solver also cannot do anything harmful while waiting for the next restart (but for this to hold it is crucial that the solver never deletes any learned clauses).

It is still mostly an open problem to establish theoretical results about how the commonly used CDCL heuristics contribute (or sometimes fail to contribute) to the success of CDCL search. This is challenging to do from a mathematical point of view, however, since these heuristics are quite sophisticated and often depend on the full history of the CDCL execution so far.

For clause learning, a seemingly obvious approach would be to prioritize learning short clauses, but it seems to have been known in the SAT community that

this does not work so well in practice. A theoretical explanation for this was given in [BJ10], where it was shown that for formulas that require resolution proofs with large clauses CDCL solvers also need to learn and keep large clauses in order to run fast (i.e., it is not enough that large clauses appear as intermediate steps in the conflict analysis). Regarding clause deletion, the theoretical results in Sections 7.4.3 and 7.4.4 can be interpreted as saying that the quite aggressive deletion policies in modern CDCL solvers can incur a substantial (sometimes exponential) increase in running time, although this can only be rigorously proven for specially designed theoretical benchmark formulas. As to activity-related decision heuristics such as VSIDS [MMZ+01], VMTF [Rya04], or phase saving [PD07], there is very little by way of concrete theoretical results.

Open Problem 7.2. Prove simulation or separation results for resolution versus CDCL when the solver is constrained to use decision strategies such as VSIDS, VMTF, and/or phase saving.

A very recent paper [Vin20] shows, somewhat informally speaking, that there are unsatisfiable CNF formulas for which standard decision heuristics such as VSIDS and VMTF can be exponentially worse than optimal decisions. Another recent contribution [MPR20] studies the power of CDCL when the decision heuristic is replaced by a fixed ordering of the variables, but the strongest results are unfortunately for a model that is fairly far from practice when it comes to modelling unit propagation and clause learning.

An even more fundamental open problem is to understand the role of restarts for CDCL solvers and whether they add to the theoretical reasoning power. Phrased in theoretical language, the question is whether restarts are really necessary in order for CDCL to be able to simulate resolution, or whether this can be done also without restarts. To conclude this section, we briefly discuss two related approaches to studying this problem by modelling CDCL without restarts as a proof system, namely *pool resolution* [Van05] and *RegWRTI* [BHJ08].

A pool resolution refutation consists of a resolution refutation π with an associated regular depth-first traversal; in other words, there is a depth-first traversal of the DAG G_π such that at each point during the traversal the path from the current clause back to the empty clause does not resolve more than once on any given variable. The intuition is that a regular depth-first traversal corresponds to the CDCL search, so if clauses are learned as they are traversed, they do not need to be traversed again. The fact that CDCL never chooses a decision literal which is already set and that clause learning only learns clauses containing negated literals imposes a regularity condition for each path in the depth-first search. One aspect in which pool resolution seems stronger than CDCL, though, is that the depth-first traversal allows a much richer set of clauses to be "learned" than those derived by trivial resolution from the current clause database.

The RegWRTI system is similar in spirit to pool resolution, but it uses a restricted form of clause learning that (unlike pool resolution) allows learning only clauses that can be learned using cuts in a conflict graph. Thus RegWRTI better approximates the way clause learning works in CDCL solvers. The definition of RegWRTI is a bit complicated, so we omit it here. It is open whether pool resolution or RegWRTI simulate resolution; indeed, several candidates for separation

have failed to give a separation [BB12, BBJ14, BK14].

Pool resolution and RegWRTI do not fully capture CDCL without restarts, as they do not incorporate *self-subsumption* [SB09, HS09] (also called *clause minimization*) during clause learning. For an example of self-subsumption, see Figure 7.4, where the 1UIP clause $\overline{a} \vee \overline{b} \vee \overline{c} \vee \overline{y}$ can be resolved with the clause $\overline{a} \vee \overline{b} \vee c$ to yield a better learned clause $\overline{a} \vee \overline{b} \vee \overline{y}$. Since this resolves on the literal c even though c has been set false, it does not fit the framework of pool resolution or RegWRTI. It is possible that pool resolution or especially RegWRTI can simulate clause learning augmented with self-subsumption, but this is an open question.

Open Problem 7.3. Can CDCL as modelled in [AFT11, PD11, EJL$^+$16], but without restarts, efficiently simulate resolution? Or is it possible to prove that resolution is stronger than such a model of CDCL, or even stronger than RegWRTI or pool resolution?

7.4. Resolution and Proof Complexity

The previous section described the close connections between the resolution proof system and SAT solvers based on conflict-driven clause learning. Thanks to these connections, one way of analysing the potential and limitations of CDCL proof search is to study resolution refutations and establish upper and lower bounds on the complexity of such refutations. A lower bound on resolution proof length gives a corresponding lower bound on CDCL execution time, since the proof search cannot run faster than the smallest resolution proof it could possibly find.[14] Conversely, an upper bound on resolution refutation length points to at least the possibility of good performance by CDCL search algorithms. There are other ways to measure resolution proof complexity beside sheer length. Notably, bounding the width or space of resolution refutations can shed light on the effectiveness of different clause learning and clause erasure strategies. In this section, we review what is known about these different proof complexity measures for resolution.

7.4.1. Resolution Length

We start by recalling that the *length* (also referred to as the *size*) of a resolution refutation is the number of clauses in it, where the clauses are counted with repetitions (which is relevant for tree-like or regular refutations). In general, proof length/size is the most fundamental measure in proof complexity, and as just discussed, lower bounds for resolution length imply lower bounds on CDCL solver running time.

Any CNF formula of size N can be refuted in resolution in length $\exp(\mathrm{O}(N))$, and there are formulas for which matching $\exp(\Omega(N))$ lower bounds are known. Let us discuss some examples of formulas known to be hard with respect to resolution length.

Our first example is the ***pigeonhole principle (PHP)***, which says that "m pigeons do not fit into n holes without sharing holes if $m > n$." This is

[14] As already discussed, however, this does not take into account the effect of preprocessing techniques that cannot be formalized within the resolution proof system, but it is still the case that resolution lower bounds often imply hardness also in practice.

arguably the single most studied combinatorial principle in all of proof complexity (see [Raz02] for a survey). When written as an unsatisfiable CNF formula, this becomes the claim that, on the contrary, $m > n$ pigeons do fit into n holes. To encode this, one uses variables $p_{i,j}$ to denote "pigeon i goes into hole j," and write down the following clauses, where $i \neq i'$ range over $1, \ldots, m$ and $j \neq j'$ range over $1, \ldots, n$:

$$p_{i,1} \vee p_{i,2} \vee \cdots \vee p_{i,n} \qquad \text{[every pigeon } i \text{ gets a hole]} \tag{7.7a}$$

$$\overline{p}_{i,j} \vee \overline{p}_{i',j} \qquad \text{[no hole } j \text{ gets two pigeons } i \neq i'\text{]} \tag{7.7b}$$

There are also variants where one in addition has "functionality" and/or "onto" axioms

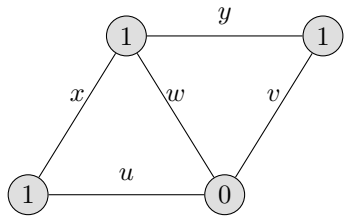
$$\overline{p}_{i,j} \vee \overline{p}_{i,j'} \qquad \text{[no pigeon } i \text{ gets two holes } j \neq j'\text{]} \tag{7.7c}$$

$$p_{1,j} \vee p_{2,j} \vee \cdots \vee p_{m,j} \qquad \text{[every hole } j \text{ gets a pigeon]} \tag{7.7d}$$

In a breakthrough result, Haken [Hak85] proved that the PHP formulas consisting of clauses (7.7a) and (7.7b) require length $\exp(\Omega(n))$ in resolution for $m = n + 1$ pigeons, and his proof can be extended to work also for the ***onto FPHP formulas*** consisting of all clauses (7.7a)–(7.7d).[15] Later work [Raz04a, Raz03, Raz04b] has shown that all of the PHP formula variants remain hard even for arbitrarily many pigeons m, requiring resolution length $\exp\bigl(\Omega\bigl(n^{\delta}\bigr)\bigr)$ for some $\delta > 0$ — such formulas with $m \gg n$ are referred to as ***weak pigeonhole principle (WPHP) formulas***, since the claim made by the pigeonhole principle gets weaker and weaker as m increases compared to n. What all of these lower bounds mean, intuitively, is that the resolution proof system really cannot count: even faced with the preposterous claim that infinitely many pigeons can be mapped in a one-to-one fashion into a some finite number n of holes, resolution cannot refute this claim with a proof of length polynomially bounded in the number of holes.

Since pigeonhole principle formulas have size $N = \Theta\bigl(n^3\bigr)$, Haken's lower bound is only of the form $\exp\bigl(\Omega\bigl(\sqrt[3]{N}\bigr)\bigr)$ expressed in terms of formula size, however, and so does not quite match the $\exp(\mathrm{O}(N))$ worst-case upper bound. The first truly exponential lower bound on length was obtained for ***Tseitin formulas*** (an example of which is shown in Figure 7.9), which encode (the negation of) the principle that "the sum of the vertex degrees in a graph is even." Here the variables correspond to the edges in an undirected graph G of bounded degree. Every vertex in G is labelled 0 or 1 so that the sum of the vertex labels is odd. The Tseitin formula for G is the CNF formula which is the conjunction of the set of clauses expressing that for each vertex of G the parity of the number of true edges incident to that vertex is equal to the vertex label. See Figure 7.9b, which displays the formula corresponding to the labelled graph in Figure 7.9a.

[15] To generate these formulas in the standard DIMACS format used by SAT solvers, one can use the tool *CNFgen* [LENV17, CNF] with the command line `cnfgen php` ⟨`m`⟩ ⟨`n`⟩ to obtain PHP formulas with m pigeons and n holes, `cnfgen php --functional` ⟨`m`⟩ ⟨`n`⟩ for FPHP formulas with axioms (7.7c) also, `cnfgen php --onto` ⟨`m`⟩ ⟨`n`⟩ for onto PHP formulas with axioms (7.7d), and `cnfgen php --onto --functional` ⟨`m`⟩ ⟨`n`⟩ for onto FPHP formulas with all axioms clauses (7.7a)–(7.7d) discussed above. (Here and in all following *CNFgen* examples, the syntax ⟨`n`⟩ means that an actual value of n, without brackets, should be provided on the command line.)



(a) Graph with odd labelling.

$$
\begin{aligned}
&(x \vee u) && \wedge\ (y \vee v)\\
\wedge\ &(\overline{x} \vee \overline{u}) && \wedge\ (\overline{y} \vee \overline{v})\\
\wedge\ &(x \vee w \vee y) && \wedge\ (\overline{u} \vee \overline{w} \vee \overline{v})\\
\wedge\ &(\overline{x} \vee \overline{w} \vee y) && \wedge\ (\overline{u} \vee w \vee v)\\
\wedge\ &(\overline{x} \vee w \vee \overline{y}) && \wedge\ (u \vee \overline{w} \vee v)\\
\wedge\ &(x \vee \overline{w} \vee \overline{y}) && \wedge\ (u \vee w \vee \overline{v})
\end{aligned}
$$

(b) Corresponding Tseitin formula.

Figure 7.9: Example Tseitin formula.

If we sum over all vertices, the number of all true incident edges should be odd by the construction of the labelling. However, since such a sum counts each edge exactly twice it has to be even. Thus, the Tseitin formulas are indeed unsatisfiable. Urquhart [Urq87] established that Tseitin formulas require resolution length $\exp\bigl(\Omega\bigl(N\bigr)\bigr)$ if the underlying graph is a well-connected so-called *expander graph*. We cannot discuss the fascinating theory of expander graphs here, and instead refer to [HLW06] for more information, but suffice it to say that, e.g., a randomly sampled regular graph is an excellent expander *asymptotically almost surely*, i.e., with overwhelmingly large probability,[16] and there are also explicit constructions. Intuitively, the lower bound in [Urq87] shows that not only is resolution unable to count efficiently in general, but it cannot even do so mod 2.[17]

Another example of exponentially hard formulas are ***random k-CNF formulas*** which are generated by randomly sampling $\Delta \cdot n$ k-clauses over n variables for some large enough constant Δ depending on k. For instance, $\Delta \gtrsim 4.5$ is sufficient to get unsatisfiable 3-CNF formulas asymptotically almost surely [DBM00] (see also Chapter 7.11 on random satisfiability in this handbook). Chvátal and Szemerédi [CS88] established that resolution requires length $\exp\bigl(\Omega\bigl(N\bigr)\bigr)$ to refute such formulas (again asymptotically almost surely).[18]

By now strong lower bounds have been shown for formulas encoding ***tiling problems*** [Ale04, DR01], ***k-colourability*** [BCMM05], ***independent set*/*clique***, and ***vertex cover*** [BIS07], and many other combinatorial principles. For clique formulas there is an interesting range of parameters where lower bounds are *not* known, however, and so we discuss this family of formulas next.

[16] Formally, we say that a sequence of events $\mathcal{P}_n$ happen *asymptotically almost surely* (sometimes also referred to as *with high probability*) if $\lim_{n\to\infty} \Pr[\mathcal{P}_n] = 1$.

[17] To generate unsatisfiable Tseitin formulas in DIMACS format for random 4-regular graphs over n vertices (which yields formulas that are exponentially hard for resolution asymptotically almost surely) one can use *CNFgen* with the command line `cnfgen tseitin` ⟨`n`⟩ (or, more generally, `cnfgen tseitin` ⟨`n`⟩ ⟨`d`⟩ for d-regular graphs over n vertices as long as $d \cdot n$ is even). For a standard CDCL solver without Gaussian reasoning, Tseitin formulas over random 4-regular graphs become quite hard already for around 50 vertices.

[18] *CNFgen* generates random k-CNF formulas with m clauses over n variables with the command line `cnfgen randkcnf <k> <n> <m>`. Random 3-CNF formulas with n variables and $\Delta \cdot n$ clauses for $\Delta = 4.5$ are noticeably hard for CDCL solvers by the time $n = 350$ or so.

Given a positive integer k and a graph $G = (V, E)$ with vertices V and edges E, the clique formula encodes the claim that G has a k-clique. We have variables $x_{v,i}$ with the intended meaning "vertex v is the ith member of the clique" and also think of the vertices as $V = \{1, 2, \ldots, n\}$ so that we can list them in order and compare them. Letting u, v range over vertices and i, j range over clique membership indices, the formula consists of the following clauses:

$$x_{1,i} \vee x_{2,i} \vee \cdots \vee x_{n,i} \qquad i \in [k], \tag{7.8a}$$

$$\overline{x}_{u,i} \vee \overline{x}_{v,i} \qquad u, v \in V,\ u < v,\ i \in [k] \tag{7.8b}$$

$$\overline{x}_{v,i} \vee \overline{x}_{v,j} \qquad v \in V,\ i, j \in [k],\ i < j \tag{7.8c}$$

$$\overline{x}_{u,i} \vee \overline{x}_{v,j} \qquad (u, v) \notin E,\ i, j \in [k],\ i \neq j \tag{7.8d}$$

$$\overline{x}_{u,j} \vee \overline{x}_{v,i} \qquad u, v \in V,\ u < v,\ i, j \in [k],\ i < j \tag{7.8e}$$

Clauses (7.8a), (7.8b), and (7.8c) just encode that there is precisely one vertex chosen to be the ith clique member. Clauses (7.8d) enforce the constraint that there must be edges between all vertices in the clique. Finally, the clauses (7.8e) remove a potential (and artificial) source of hardness by specifying that the vertices $v_1, v_2, \ldots, v_k$ chosen as clique members $1, 2, \ldots, k$ should be listed in increasing order.

From a proof complexity point of view, clique formulas become interesting when the graph G does *not* have a k-clique but when this is hard to prove. This problem can clearly be solved in time roughly n^k simply by checking if any of the $\binom{n}{k}$ many sets of vertices of size k forms a clique. Such a simple approach can be formalized already in tree-like resolution, and yields refutations of polynomial length if k is constant. A popular conjecture is that the k-clique problem must require time $n^{\Omega(k)}$ in the worst case, or even on average when graphs are sampled at random (e.g., according to the Erdős-Rényi distribution where for an n-vertex graph every single edge out of the $\binom{n}{2}$ possible ones is included uniformly and independently at random with some appropriate probability p, called the edge density).[19] A natural question is whether it is possible to prove such lower bounds unconditionally for proof systems that are strong enough to capture algorithmic methods used in practice.

For tree-like resolution, an optimal $n^{\Omega(k)}$ length lower bound (i.e., optimal up to constant factors in the exponent) was established in [BGL13] for k-clique formulas on Erdős-Rényi random graphs. When k is chosen very large, on the order of n^γ for some large enough γ, $0 < \gamma < 1$, it was shown in [BIS07, Pan19] that resolution requires length scaling like $\exp\bigl(n^\delta\bigr)$ for some $\delta > 0$ for Erdős-Rényi random graphs. For smaller k, an optimal $n^{\Omega(k)}$ length lower bound was obtained for the restricted subsystem of *regular resolution* in [ABdR$^+$18], but for general resolution it remains an open problem even to establish worst-case lower bounds for small k

[19] To generate k-clique formulas over n-vertex random graphs for small values of k that seem likely to require resolution refutations of length $n^{\Omega(k)}$, one can use *CNFgen* with the command line `cnfgen kclique ⟨k⟩ gnp ⟨n⟩ ⟨p⟩` for $p = n^{-2/(k-2)}$, which is below the threshold edge density $n^{-2/(k-1)}$ for the appearance of k-cliques and hence will yield unsatisfiable formulas asymptotically almost surely. A concrete setting of parameters that seem to yield formulas with hardness that scales nontrivially with the number of vertices n in practice for CDCL solvers is $k = 10$ and $p = n^{-1/4}$.

$$\begin{pmatrix}
1&1&0&1&0&0&0&1&0&0&0\\
0&1&1&0&1&0&0&0&1&0&0\\
0&0&1&1&0&1&0&0&0&1&0\\
0&0&0&1&1&0&1&0&0&0&1\\
1&0&0&0&1&1&0&1&0&0&0\\
0&1&0&0&0&1&1&0&1&0&0\\
0&0&1&0&0&0&1&1&0&1&0\\
0&0&0&1&0&0&0&1&1&0&1\\
1&0&0&0&1&0&0&0&1&1&0\\
0&1&0&0&0&1&0&0&0&1&1\\
1&0&1&0&0&0&1&\mathbf{1}&0&0&1
\end{pmatrix}$$

(a) Matrix with row and column constraints.

$$\begin{aligned}
&(x_{1,1} \vee x_{1,2} \vee x_{1,4})\\
\wedge\ &(x_{1,1} \vee x_{1,2} \vee x_{1,8})\\
\wedge\ &(x_{1,1} \vee x_{1,4} \vee x_{1,8})\\
\wedge\ &(x_{1,2} \vee x_{1,4} \vee x_{1,8})\\
&\ \ \vdots\\
\wedge\ &(\overline{x}_{4,11} \vee \overline{x}_{8,11} \vee \overline{x}_{10,11})\\
\wedge\ &(\overline{x}_{4,11} \vee \overline{x}_{8,11} \vee \overline{x}_{11,11})\\
\wedge\ &(\overline{x}_{4,11} \vee \overline{x}_{10,11} \vee \overline{x}_{11,11})\\
\wedge\ &(\overline{x}_{8,11} \vee \overline{x}_{10,11} \vee \overline{x}_{11,11})
\end{aligned}$$

(b) Cardinality constraints in CNF.

Figure 7.10: Matrix and (fragment of) corresponding subset cardinality formula.

Open Problem 7.4. Prove average-case resolution length lower bounds $n^{\Omega(k)}$ for k-clique formulas over Erdős-Rényi random graphs with appropriate parameters, or worst-case length lower bounds for any family of graphs.

This problem is also of intrinsic interest in proof complexity, since these formulas do not seem to be amenable to the standard techniques for proving resolution lower bounds such as the *interpolation method* [Kra97, Pud97], *random restrictions* [BP96b], or the *length-width lower bound* in [BW01]. We will discuss some of these techniques later in the survey.

We conclude our discussion of resolution length by mentioning one slightly more recent addition to the long list of challenging combinatorial formulas already mentioned, namely the ***subset cardinality formulas*** studied in [Spe10, VS10, MN14] (also known as ***zero-one design*** or ***sgen*** formulas).

To construct these formulas, we start with an $n \times n$ $(0,1)$-matrix with 4 non-zero entries in each row and column except that one extra non-zero entry is added to some empty cell (as in Figure 7.10a, where the extra 1 in the bottom row is in bold face). The variables of the formula are the non-zero entries of the matrix, yielding a total of $4n+1$ variables. For each row of 4 ones in the matrix, we write down the natural 3-CNF formula encoding the *positive cardinality constraint* that at least 2 variables must be true (as in the first set of clauses in Figure 7.10b), and for the row with 5 ones the 3-CNF formula encoding that a strict majority of 3 variables must be true. For the columns we instead encode *negative cardinality constraints* that a majority of the variables in each column must be false (see the last set of clauses in Figure 7.10b). The formula consisting of the conjunction of all these clauses must be unsatisfiable, since a strict majority of the variables cannot be true and false simultaneously. We will have reason to return to these formulas below when we discuss connections between CDCL and resolution, and also when discussing cutting planes and pseudo-Boolean solving.

It was shown empirically in [Spe10, VS10] that these formulas are very challenging for CDCL solvers, but there was no analysis of the theoretical hardness. Such an analysis was provided by [MN14], where it was established that subset

cardinality formulas are indeed exponentially hard if the underlying matrix is an expander (informally, if every small-to-medium set of rows has non-zero entries in many distinct columns).[20]

7.4.2. Resolution Width

A second complexity measure in resolution, which is almost as well studied as length, is the *width* of refutations measured as the size of a largest clause in a resolution refutation. It is clear that the width needed to refute a formula is never larger than the number of variables n, which is in turn less than the total formula size N. It is also easy to see that an upper bound w on resolution width implies an upper bound $O((3n)^w)$ on resolution length, simply because the total number of distinct clauses of width at most w over n variables is less than $(3n)^w$. Incidentally, this simple counting argument turns out to be essentially tight, in that there are k-CNF formulas refutable in width w that require resolution length $n^{\Omega(w)}$, as shown in [ALN16].

Much less obviously, however, and much more interestingly, strong enough width lower bounds imply strong length lower bounds. Ben-Sasson and Wigderson [BW01] (using methods based on [CEI96, IPS99]) showed that for a k-CNF formula over n variables it holds that

$$\text{refutation length} \geq \exp\left(\Omega\left(\frac{(\text{refutation width} - k)^2}{n}\right)\right) \tag{7.9}$$

where "refutation length" and "refutation width" mean the minimum length and minimum width, respectively, of any resolution refutations of the formula. (Note that these two minima could potentially be realized by different refutations.) The inequality (7.9) implies that if one can prove that a formula requires width $\omega\big(\sqrt{n \log n}\big)$, this immediately yields a superpolynomial length lower bound, and a width lower bound $\Omega(N)$ in terms of the formula size N (which is lower-bounded by the number of variables n) implies a truly exponential $\exp(\Omega(N))$ length lower bound. Almost all known lower bounds on resolution length can be derived via width lower bounds in this way (in particular, essentially all the bounds discussed in Section 7.4.1[21] although the ones predating [BW01] were originally not obtained in this way).

For tree-like resolution refutations of k-CNF formulas, the paper [BW01] proved a sharper version

$$\text{tree-like refutation length} \geq 2^{\text{refutation width}-k} \tag{7.10}$$

of the bound in (7.9) for general resolution. This means that for tree-like resolution, even width lower bounds $\omega\big(\log N\big)$ yield superpolynomial length lower

[20] *CNFgen* generates subset cardinality formulas for random $n \times n$ matrices — which are expanding, and so yield formulas that are exponentially hard for resolution, asymptotically almost surely — with the command line `cnfgen subsetcard <n>`. These formulas get noticeably hard for CDCL solvers around $n = 25$.

[21] To be precise, the exceptions are [Ale04, DR01, Raz04a, Raz03, Raz04b], where the number of variables n, and hence the formula size N, is at least as large as the refutation width squared, and where other methods must therefore be used to prove lower bounds on resolution length.

bounds. For general resolution, however, a width lower bound even as large as $\Omega\big(\sqrt{n \log n}\big)$ does not imply any length lower bound according to (7.9). This raises the question of whether it is possible to improve the analysis so that (7.9) can be strengthened to something closer to (7.10) also for general resolution. Bonet and Galesi [BG01] showed that this is not the case by studying another interesting combinatorial benchmark formula, which we describe next.

The ***ordering principle*** says that "every finite (partially or totally) ordered set $\{e_1, \ldots, e_n\}$ has a minimal element." To encode the negation of this statement in CNF, we use variables $x_{i,j}$ to denote "$e_i < e_j$" and write down the following clauses (for $i \neq j \neq k \neq i$ ranging over $1, ..., n$):

$$\overline{x}_{i,j} \vee \overline{x}_{j,i} \qquad \text{[anti-symmetry; not both } e_i < e_j \text{ and } e_j < e_i\text{]} \tag{7.11a}$$

$$\overline{x}_{i,j} \vee \overline{x}_{j,k} \vee x_{i,k} \qquad \text{[transitivity; } e_i < e_j \text{ and } e_j < e_k \text{ implies } e_i < e_k\text{]} \tag{7.11b}$$

$$\bigvee\nolimits_{1 \leq i \leq n,\, i \neq j} x_{i,j} \qquad \text{[} e_j \text{ is not a minimal element]} \tag{7.11c}$$

One can also add axioms

$$x_{i,j} \vee x_{j,i} \qquad \text{[totality; either } e_i < e_j \text{ or } e_j < e_i\text{]} \tag{7.11d}$$

to specify that the ordering has to be total.[22] This yields a formula over $\Theta\big(n^2\big)$ variables of total size $N = \Theta\big(n^3\big)$. We remark that variants of ordering principle formulas also appear under the names ***least number principle formula*** and ***graph tautology formula*** in the literature.

It was conjectured in [Kri85] that these formulas should be exponentially hard for resolution, but Stålmarck [Stå96] showed that they are refutable in length $\mathrm{O}(N)$ (even without the clauses (7.11d)).

As the formula is described above, it does not really make sense to ask about the refutation width, since already the axiom clauses (7.11c) have unbounded width. However, one can convert the formula to 3-CNF by applying the transformation from (7.1a) to (7.1b) to the wide axioms (7.11c), and for this version of the formula [BG01] established a width lower bound $\Omega\big(\sqrt[3]{N}\big)$ (which is tight, and holds even if the axioms (7.11d) are also added). This shows that even polynomially large resolution width does not necessarily imply any length lower bounds for general resolution (and, in view of (7.10), it provides an exponential separation in proof power between general and tree-like resolution, although even stronger separations are known [BIW04]).

7.4.3. Resolution Space

The study of the space complexity of proofs, which was initiated in the late 1990s, was originally motivated by considerations of SAT solver memory usage, but has also turned out to be of intrinsic interest for proof complexity. Space can be measured in different ways — here we focus on the most well studied measure of *clause space*, which is the maximal number of clauses needed in memory while

[22] *CNFgen* generates the ordering principle formula for a set of n elements with the command line `cnfgen op` ⟨`n`⟩ and also adds totality axioms with `cnfgen op --total` ⟨`n`⟩. These formulas are easy in practice for CDCL solvers that use VSIDS with a small enough decay factor or VMTF.

verifying the correctness of a resolution refutation.[23] Thus, in what follows below "space" will always mean "clause space."

The space usage of a resolution refutation at step t is the number of clauses at steps $\leq t$ that are used at steps $\geq t$. Returning to our example resolution refutation in Figure 7.1, the space usage at step 8 is 5 (the clauses in memory at this point are clauses 1, 2, 6, 7, and 8). The space of a proof is obtained by measuring the space usage at each step in the proof and taking the maximum. Phrased differently, one can view the formula as being stored in read-only input memory, from where the axiom clauses can be read into working memory. The resolution rule can only be applied to clauses currently in working memory, and if a clause has been erased from working memory, then it is gone, and will have to be rederived if it is to be used again (or read again from the read-only input, in the case of axiom clauses). Then space measures how many clauses are used in working memory to perform the resolution refutation. Incidentally, it is not hard to see that the proof in Figure 7.1 is not optimal when it comes to minimizing space. We could do the same refutation in space 4 instead by processing the clauses in the order 4, 5, 6, 3, 7, 2, 8, 1, 9, 10. (In fact, it is even conceivable that if minimizing space is all we care about, then it might be beneficial to forget clauses and rederive them later, even if it means repeating the same steps in the resolution refutation multiple times. This indeed turns out to be the case, as discussed in Section 7.4.4 below.)

Perhaps somewhat surprisingly, any unsatisfiable CNF formula of size N can always be refuted in resolution space at most $N + \mathrm{O}(1)$ as shown by [ET01],[24] though the resolution refutation thus obtained might have exponential length. Lower bounds on space were shown for pigeonhole principle formulas and Tseitin formulas in [ABRW02, ET01] and for random k-CNF formulas in [BG03]. For the latter two formula families the (optimal linear) lower bounds matched exactly previously known width lower bounds, and also the proof techniques had a very similar flavour. This led to the question of whether there was some deeper connection waiting to be discovered. In a very elegant paper, Atserias and Dalmau [AD08] confirmed this by showing that the inequality

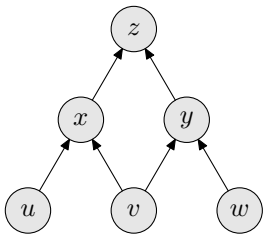
$$\text{refutation space} \geq \text{refutation width} - k \tag{7.12}$$

holds for resolution refutations of k-CNF formulas. The proof of (7.12) is beautiful but uses a somewhat non-explicit argument based on finite model theory. A more explicit proof, which works by simple syntactic manipulations to construct a small-width refutation from a small-space refutation, was presented in [FLM$^+$15].

Since for all formulas studied up to [AD08] the width and space complexity measures turned out to actually coincide, it is natural to ask whether (7.12)

[23] Note, though, that this measure underestimates the actual memory usage, since storing a clause requires more than a constant amount of memory (this is similar to how we ignore the size of clauses when defining the size of resolution proofs to be the total number of clauses in the proof). For completeness, we mention that there is also a measure *total space*, counting the total number of literals in memory (with repetitions), which has been studied in, e.g., [ABRW02, BGT14, BBG$^+$17, Bon16, GKT19].

[24] This space upper bound can also be obtained simply by running CDCL (or even DPLL) as described in Section 7.3.1 with some arbitrary but fixed variable ordering, with the (non-standard) modification that after each backjump all clauses that are no longer needed to explain the propagation of literals in the current assignment are immediately erased.



(a) Pyramid graph Π_2 of height 2.

$$
\begin{aligned}
& u \\
\wedge\ & v \\
\wedge\ & w \\
\wedge\ & (\overline{u} \vee \overline{v} \vee x) \\
\wedge\ & (\overline{v} \vee \overline{w} \vee y) \\
\wedge\ & (\overline{x} \vee \overline{y} \vee z) \\
\wedge\ & \overline{z}
\end{aligned}
$$

(b) Pebbling contradiction Peb_{Π_2}.

Figure 7.11: Example pebbling contradiction for the pyramid of height 2.

can be strengthened to an asymptotic equality. The answer to this question is negative. As shown in the sequence of works [Nor09a, NH13, BN08], there are formulas that can be refuted in width $\mathrm{O}(1)$ and length $\mathrm{O}(N)$ but require space $\Omega(N/\log N)$ (i.e., formulas that are maximally easy for width but exhibit worst-case behaviour for space except for a log factor, and this result is tight since it can be shown (using a result from [HPV77] that is, incidentally, very related to the next topic of *pebbling*) that any formula refutable in length $\mathrm{O}(N)$ can also be refuted in space $\mathrm{O}(N/\log N)$).

The formulas used to obtain the separation result just described are ***pebbling contradictions*** (also called ***pebbling formulas***) encoding so-called *pebble games* on bounded fan-in DAGs, which for the purposes of this discussion we additionally require to have a unique sink. In the "vanilla version" of the formula (illustrated in Figure 7.11), there is one variable associated to each vertex and clauses encoding that

- the source vertices are all true;
- if all immediate predecessors are true, then the successor vertex is true;
- but the sink is false.

There is an extensive literature on pebbling space and time-space trade-offs from the 1970s and 80s, with Pippenger [Pip80] and Savage [Sav98, Chapter 10] giving excellent overviews of some classic results in the area. Some more recent developments are covered in the upcoming survey [Nor20]. Pebbling contradictions have been useful before in proof complexity in various contexts, e.g., in [RM99, BEGJ00, BW01]. Since pebbling contradictions can be shown to be refutable in constant width but there are graphs for which the pebble game requires large space, one could hope that the pebbling properties of such DAGs would somehow carry over to resolution refutations of pebbling formulas and help us separate space and width.

Unfortunately, this hope cannot possibly materialize — a quick visual inspection of Figure 7.11b reveals that this is a Horn formula (i.e., having at most one positive literal in each clause), and such formulas are maximally easy for length,

$$
\begin{array}{ll}
(u_1 \vee u_2) & \wedge\ (v_1 \vee \overline{v}_2 \vee \overline{w}_1 \vee w_2 \vee y_1 \vee y_2) \\
\wedge\ (\overline{u}_1 \vee \overline{u}_2) & \wedge\ (v_1 \vee \overline{v}_2 \vee \overline{w}_1 \vee w_2 \vee \overline{y}_1 \vee \overline{y}_2) \\
\wedge\ (v_1 \vee v_2) & \wedge\ (\overline{v}_1 \vee v_2 \vee w_1 \vee \overline{w}_2 \vee y_1 \vee y_2) \\
\wedge\ (\overline{v}_1 \vee \overline{v}_2) & \wedge\ (\overline{v}_1 \vee v_2 \vee w_1 \vee \overline{w}_2 \vee \overline{y}_1 \vee \overline{y}_2) \\
\wedge\ (w_1 \vee w_2) & \wedge\ (\overline{v}_1 \vee v_2 \vee \overline{w}_1 \vee w_2 \vee y_1 \vee y_2) \\
\wedge\ (\overline{w}_1 \vee \overline{w}_2) & \wedge\ (\overline{v}_1 \vee v_2 \vee \overline{w}_1 \vee w_2 \vee \overline{y}_1 \vee \overline{y}_2) \\
\wedge\ (u_1 \vee \overline{u}_2 \vee v_1 \vee \overline{v}_2 \vee x_1 \vee x_2) & \wedge\ (x_1 \vee \overline{x}_2 \vee y_1 \vee \overline{y}_2 \vee z_1 \vee z_2) \\
\wedge\ (u_1 \vee \overline{u}_2 \vee v_1 \vee \overline{v}_2 \vee \overline{x}_1 \vee \overline{x}_2) & \wedge\ (x_1 \vee \overline{x}_2 \vee y_1 \vee \overline{y}_2 \vee \overline{z}_1 \vee \overline{z}_2) \\
\wedge\ (u_1 \vee \overline{u}_2 \vee \overline{v}_1 \vee v_2 \vee x_1 \vee x_2) & \wedge\ (x_1 \vee \overline{x}_2 \vee \overline{y}_1 \vee y_2 \vee z_1 \vee z_2) \\
\wedge\ (u_1 \vee \overline{u}_2 \vee \overline{v}_1 \vee v_2 \vee \overline{x}_1 \vee \overline{x}_2) & \wedge\ (x_1 \vee \overline{x}_2 \vee \overline{y}_1 \vee y_2 \vee \overline{z}_1 \vee \overline{z}_2) \\
\wedge\ (\overline{u}_1 \vee u_2 \vee v_1 \vee \overline{v}_2 \vee x_1 \vee x_2) & \wedge\ (\overline{x}_1 \vee x_2 \vee y_1 \vee \overline{y}_2 \vee z_1 \vee z_2) \\
\wedge\ (\overline{u}_1 \vee u_2 \vee v_1 \vee \overline{v}_2 \vee \overline{x}_1 \vee \overline{x}_2) & \wedge\ (\overline{x}_1 \vee x_2 \vee y_1 \vee \overline{y}_2 \vee \overline{z}_1 \vee \overline{z}_2) \\
\wedge\ (\overline{u}_1 \vee u_2 \vee \overline{v}_1 \vee v_2 \vee x_1 \vee x_2) & \wedge\ (\overline{x}_1 \vee x_2 \vee \overline{y}_1 \vee y_2 \vee z_1 \vee z_2) \\
\wedge\ (\overline{u}_1 \vee u_2 \vee \overline{v}_1 \vee v_2 \vee \overline{x}_1 \vee \overline{x}_2) & \wedge\ (\overline{x}_1 \vee x_2 \vee \overline{y}_1 \vee y_2 \vee \overline{z}_1 \vee \overline{z}_2) \\
\wedge\ (v_1 \vee \overline{v}_2 \vee w_1 \vee \overline{w}_2 \vee y_1 \vee y_2) & \wedge\ (z_1 \vee \overline{z}_2) \\
\wedge\ (v_1 \vee \overline{v}_2 \vee w_1 \vee \overline{w}_2 \vee \overline{y}_1 \vee \overline{y}_2) & \wedge\ (\overline{z}_1 \vee z_2)
\end{array}
$$

Figure 7.12: Pebbling contradiction in Figure 7.11b with XOR-substitution.

width, and space since they are decided by unit propagation. However, we can modify these formulas by substituting for every variable x an exclusive or $x_1 \oplus x_2$ of two new variables, and then expand to CNF in the canonical way to get a new formula. This process is called *XOR-ification* or *XOR-substitution* and is perhaps easiest to explain by example. Performing this substitution in the clause

$$\overline{x} \vee y \tag{7.13a}$$

we obtain the formula

$$\neg(x_1 \oplus x_2) \vee (y_1 \oplus y_2)\ , \tag{7.13b}$$

which when expanded out to CNF becomes

$$
\begin{array}{l}
(x_1 \vee \overline{x}_2 \vee y_1 \vee y_2) \\
\wedge\ (x_1 \vee \overline{x}_2 \vee \overline{y}_1 \vee \overline{y}_2) \\
\wedge\ (\overline{x}_1 \vee x_2 \vee y_1 \vee y_2) \\
\wedge\ (\overline{x}_1 \vee x_2 \vee \overline{y}_1 \vee \overline{y}_2)\ .
\end{array} \tag{7.13c}
$$

As another example, applying XOR-substitution to Figure 7.11b yields the formula in Figure 7.12.[25]

Using such XOR-substitution, it turns out that the pebbling contradiction inherits the time-space trade-offs of the pebbling DAG in terms of which it is defined [BN08, BN11] (and there is nothing magical with XOR — this can be shown to work also for substitution with other Boolean functions that have the

[25] To generate pebbling formulas as in Figure 7.11b for pyramid graphs of height h, run *CNFgen* with the command line `cnfgen peb pyramid` ⟨`h`⟩. Doing `cnfgen peb pyramid` ⟨`h`⟩ `-T xor 2` yields XOR-ified pebbling formulas as illustrated in Figure 7.12.

right properties). Now the strong space-width separation described above is obtained by plugging in the pebbling DAGs studied in [PTC77, GT78].

7.4.4. Resolution Trade-offs

In the preceding sections, we have seen that for all the complexity measures of length, width, and space there are formulas which are maximally hard for these measures. Suppose, however, that we are given a formula that is guaranteed to be *easy* for two or more of these measures. Can we then find a resolution refutation that optimizes these complexity measures simultaneously? Or are there trade-offs, so that minimizing one measure must cause a sharp increase in the other measure? Such questions about trade-offs have a long history in computational complexity theory, but it seems that Ben-Sasson [Ben09] was first to raise the issue in the context of proof complexity.

It should be noted that this kind of trade-off questions need to be phrased slightly more carefully in order to be really interesting. As observed in [Nor09b], it is often possible to prove trade-off results simply by gluing together two formulas F and G over disjoint sets of variables which have different proof complexity properties, and then obtain a trade-off result from the fact that any proof of unsatisfiablity has to refute either F or G. In order to eliminate such examples and obtain formulas that have inherent trade-off properties, we can additionally require that the formulas in question should be minimally unsatisfiable, i.e., that if any clause in the formula is removed, then the residual formula is satisfiable. For most of the trade-off results we consider here, the formulas are of this flavour. Also, it can be noted that for the strongest trade-off results discussed below, the trick of gluing together disjoint formulas cannot yield trade-offs with so strong parameters anyway.

The first trade-off result in proof complexity seems to have been obtained in [Ben09], where a strong space-width trade-off was established. Namely, there are formulas for which

- there are refutations in width $\mathrm{O}(1)$;
- there are also refutations in space $\mathrm{O}(1)$;
- but optimizing one measure causes (essentially) worst-case behaviour for the other measure, in that the product of the width and the space for any refutation must be $\Omega(N/\log N)$ (where N is the size of the formula, and the number of variables is also $\Theta(N)$).

This holds for the "vanilla version" of the pebbling contradictions in Figure 7.11b (if one again uses the graphs studied in [PTC77, GT78]). Using techniques from [Raz16a], this space-width trade-off was strengthened in [BN20] to show that there are formulas over n variables for which resolution refutations in width w require space almost n^w, i.e., far above the linear worst-case upper bound for space.

Regarding trade-offs between length and space, it was shown in [BN11, BBI16, BNT13] that there are formulas that can be refuted in short length and also in small space, but where even slightly optimizing one of these complexity measures causes a dramatic blow-up for the other measure. One way of obtaining such

results is to take DAGs with strong pebbling time-space trade-offs (as in, e.g., [CS80, CS82, LT82, Nor12]) and consider substituted pebbling formulas (as in Figure 7.12) over such DAGs. To give an example of such a result from [BN11], there are CNF formulas F_n of size $\Theta(n)$ and constants $\kappa' \ll \kappa$ such that:

- The formulas F_n have resolution refutations in space $\kappa' \cdot n/\log n$.
- It is also possible to refute F_n in resolution in length $\mathrm{O}(n)$ and space $\mathrm{O}(n)$ simultaneously.
- However, any resolution refutation of F_n in space at most $\kappa \cdot n/\log n$ has length $\exp\bigl(n^{\Omega(1)}\bigr)$.

Another example of formulas exhibiting length-space trade-offs are Tseitin formulas over long, narrow rectangular grids. Building on [BBI16], it was shown in [BNT13] that there are formula families $\{F_{n,w}\}$, with $1 \leq w \leq n^{1/4}$, which are of size $\Theta(n)$ and have the following properties:

- The formulas F_n have resolution refutations in space $\mathrm{O}\bigl(w \log n\bigr)$.
- It is also possible to refute F_n in length $n^{O(1)}2^w$ and space $2^w + n^{O(1)}$.
- But for any resolution refutation of F_n in space s the refutation length is lower-bounded by $\left(\frac{2^{\Omega(w)}}{s}\right)^{\Omega\left(\frac{\log\log n}{\log\log\log n}\right)}$.

One interesting aspect of this second result is that if we choose $w = \kappa \log n$ for some suitably large constant κ, then it follows that short refutations of these formulas require even superlinear space.[26] That is, although we know that any CNF formula can be refuted in linear space, short refutations can require space that is much larger than this worst-case upper bound. (It can also be noted that both of the above results are actually slightly stronger than as described here, but we omit the full technical details in order to give simpler statements.)

What a length-space trade-off result like the two examples above says is that if a resolution is of short length, then at some point during the refutation a lot of space is being used. Such results do not a priori rule out, though, that this high space usage could be an isolated spike, and that most of the time the refutation uses very little space. A more recent work [AdRNV17] established more robust trade-offs between length and *cumulative space*, exhibiting formulas where any short proof has to use a lot of space throughout the resolution refutation.

For length versus width, we know that short refutation length implies small refutation width by (7.9). The proof of this inequality works by transforming a given short refutation into a narrow one, but the length blows up exponentially in the process. Thapen [Tha16] showed that this blow-up is unavoidable by exhibiting formulas for which there exist resolution refutations in short length, but for which any refutation in width as guaranteed by (7.9) has to be exponentially long. These formulas are slightly tricky to describe, however, and so we do not do so here. A technical issue with Thapen's result is that for all other trade-offs dis-

[26] To generate such Tseitin formulas, run *CNFgen* with the command line `cnfgen tseitin randomodd grid` ⟨`N`⟩ ⟨`M`⟩ for N scaling like $\kappa \log(M)$ for some suitably large constant κ. It appears, though, that these formulas are just too hard to be solved by CDCL SAT solvers regardless of memory management, and so even though these formulas could potentially provide a time-space trade-off for SAT solvers this is more of a theoretical notion than something that could be observed in practice.

cussed above there are k-CNF formulas (for $k = \mathrm{O}(1)$) that exhibit this behaviour, but Thapen's formulas have clauses of logarithmic width. A resolution length-width trade-off result as in [Tha16] but for k-CNF formulas was recently obtained in [LNSS20]. We also want to mention in this context that in a very intriguing work Razborov [Raz16a] obtained doubly exponential length-width trade-offs in tree-like resolution (this is measured in the number of variables in the formulas, which have exponential size and polynomial width).

7.4.5. Theoretical Complexity Measures and Hardness in Practice

The next topic we wish to discuss is whether practical hardness for CDCL is in any way related to the complexity measures of resolution length, width, and space. One interesting observation in this context is that it follows from the results reviewed in Section 7.4 — if we "normalize" length by taking a logarithm, since it can be exponential in the formula size N whereas the worst-case upper bounds for width and space are linear — that for any k-CNF formula the inequalities

$$\log(\text{refutation length}) \lesssim \text{refutation width} \lesssim \text{refutation space} \tag{7.14}$$

hold. Thus, length, width, and space form an hierarchy of increasingly strict hardness measures. Let us briefly discuss the measures again in this light:

- We know that length provides a lower bound on CDCL running time[27] and that CDCL polynomially simulates resolution [PD11]. However, the results in [AM19, AR08, MPW19] suggest that short resolution proofs should be intractable to find in the worst case.
- Regarding width, searching for proofs in small width is apparently a well-known heuristic in the AI community, and [AFT11] proved that CDCL has the potential to run fast if such proofs exist. (Also, note that for formulas with resolution proofs in constant width the impossibility result in [AM19] does not apply, since for such formulas exhaustive search for small-width resolution proofs runs in polynomial time.)
- As to space, memory consumption is an important bottleneck for SAT solvers in practice, and space complexity results provide lower bounds on CDCL clause database size. One downside of this is that the bounds can be at most linear, and the solver would certainly use a linear amount of memory just to store the input. However, it is important to note that the space lower bounds can be shown to hold even in a model where we only charge for clauses in addition to the input clauses, and since it is a proof complexity lower bound it applies even to solvers which would somehow magically know *exactly* which clauses they need to keep. It could therefore be argued that in reality probably much more memory than this bare minimum should be needed.

Are width or even space lower bounds relevant indicators of CDCL hardness? Or could it be true in practice that CDCL does essentially as well as resolution with respect to length/running time? These are not mathematically well-defined

[27] Again, except if some non-resolution-based preprocessing or inprocessing techniques happen to be very successful.

questions, since state-of-the-art CDCL solvers are moving targets, but perhaps it could still possible to perform experiments and draw interesting conclusions? Such an approach was proposed in [ABLM08], and the paper [JMNŽ12] reports on what seems to have been the first systematic attempt to implement this program.

In view of the discussion above it seems too optimistic that length complexity should be a reliable indicator of CDCL hardness. The work in [JMNŽ12] was therefore focused on comparing width and space by running extensive experiments on formulas with constant width complexity (and linear length complexity) but varying space complexity to see whether running time correlated with space. These experiments produced lots of interesting data, but it seems fair to say that the results are inconclusive. For some families of formulas the correlation between running time and space complexity looks very nice, but for other formulas the results seem quite chaotic.

One comment worth making here is that the experiments in [JMNŽ12] only considered the worst-case space complexity of formulas. It might be that it would be more relevant to study formulas exhibiting length-space trade-offs as in [BN11, BBI16, BNT13] or cumulative space lower bounds as in [AdRNV17]. Another issue that should be pointed out is that formulas with low width complexity and varying space complexity are hard to find — pretty much the only known examples are the substituted pebbling formulas discussed in Section 7.4.3. Thus, it is not even clear whether the experiments measured differences in width and space complexity or some other property specific to these particular formulas. This problem seems inherent, however. One cannot just pick arbitrary benchmark formulas and compute the width and space complexity for them before running experiments, since deciding width is EXPTIME-complete [Ber12] and deciding space appears likely to be PSPACE-complete.

7.4.6. Using Theory Benchmarks to Shed Light on CDCL Heuristics

Although modern CDCL solvers are routinely used to solve real-world instances with hundreds of thousands or even millions of variables, it seems fair to say that it is still very poorly understood how these solvers can be so unreasonably effective. As should be clear from the description in Section 7.3, the basic architecture of CDCL solvers is fairly simple, but the secret behind the impressive performance of state-of-the-art solvers lies in a careful implementation of the basic CDCL algorithm with highly optimized data structures, as well in the use of dozens of sophisticated heuristics.

Unfortunately, many of these heuristics interact in subtle ways, which makes it hard to assess their relative importance. A natural approach to gain a better understanding would be to collect "real-world benchmarks" and run experiments on an instrumented CDCL solver with different parameter settings to investigate how they contribute to overall performance, as proposed in [LM02, KSM11], or even to study the resolution proofs corresponding to the solver executions [Sim14]. It seems quite tricky to implement this idea in a satisfactory way, however. The problem is that set of available benchmarks is somewhat limited, and is also a highly heterogeneous collection in terms of formula properties. For this reason it turns out to be challenging to obtain statistically significant data that would admit drawing general conclusions.

The recent paper [EGG$^+$18] instead put forth the proposition that a better understanding of CDCL could be obtained by running experiments on carefully chosen theoretical benchmarks. By tuning various parameters, one can study what impact each heuristic has on performance and how this correlates with the theoretical properties of the formulas. An obvious objection is that it is very unclear why such a study of crafted benchmarks should have any practical relevance, but some arguments in favour of this approach given in [EGG$^+$18] are as follows:

- The benchmarks are *scalable*, meaning that one can generate "the same" formula for different sizes and study how performance scales as the instance size increases. (This simple but powerful idea was perhaps first articulated clearly in an applied SAT solving setting in [PV05].)
- The benchmarks are chosen to have different *extremal* properties in a proof-complexity-theoretic sense, meaning that they can be viewed as challenging benchmarks for different heuristics for variable decisions, clause deletions, restarts, et cetera.
- Finally, in contrast to most combinatorial benchmarks traditionally used in the *SAT competitions* [SAT], which are known to be very hard for resolution, the benchmarks in [EGG$^+$18] have been constructed so as to be *easy* in the sense of having very short resolution proofs of unsatisfiability that CDCL solvers could potentially find. In view of this, it can be argued that the performance of the solver provides a measure of the quality of the proof search and how it is affected by different heuristics.

Below follow the main conclusions reported in [EGG$^+$18] after comparing the empirical results with theoretical properties of the benchmarks:

1. Learned clauses are absolutely critical for performance. It is true that the information gathered while *learning* the clauses is important for guiding other heuristics, but this is not enough — the solvers crucially need to also *store* the clauses to realize the exponential increase in reasoning power from tree-like (DPLL-style) proofs to DAG-like resolution proofs.
2. While the mathematical question of whether restarts are just a helpful heuristic or are fundamentally needed for CDCL solvers to harness the full power of resolution remains wide open, the experimental results provide some circumstantial evidence that the latter might be the case. Also, *adaptive restarts* as in [AS12] often work markedly better than the fixed-interval so-called *Luby sequence* restarts in [ES04]. More support for this observation was provided very recently in [KN20].
3. For formulas inspired by time-space trade-off results, too aggressive clause erasure can incur a stiff penalty in running time also in practice. And when memory is tight, the *literal block distance (LBD)* heuristic [AS09] often does a particularly good job at identifying useful clauses.
4. For VSIDS variable decisions [MMZ$^+$01] the choice of decay factor can sometimes be vitally important. It is not at all clear why, but one hypothesis is that this might be connected to whether the proof search needs to find DAG-like proofs or whether tree-like proofs are good enough. One can also

see that VSIDS decisions can sometimes go badly wrong for easy but tricky formulas, which suggests that there is room for further improvements in variable selection heuristics. Somewhat disappointingly, [EGG+18] did not find any support for the hypothesis that the newly proposed *learning-rate based branching (LRB)* heuristic [LGPC16] would be better than, or even distinguishable from, VSIDS.

It should perhaps be stressed that none of these findings should be considered to be a priori obvious, since proof complexity is inherently non-constructive whereas CDCL is about algorithmic proof search. Another point to emphasize is that the above findings are in no way rigorous results, but more a way of collecting circumstantial evidence for connections between theory and practice. Much work still remains to gain a more solid, mathematical understanding of when and why CDCL solvers work.

7.5. Algebraic Proof Systems

We now switch topics to *algebraic proof systems*, where formulas are translated to polynomials so that questions of satisfiability or unsatisfiability can be answered using algebraic methods of reasoning.

In what follows, we will let $\mathbb{F}$ be a field (which will usually be the finite field GF(2) with two elements $\{0,1\}$ in practical SAT solving applications, but can be any field from the point of view of proof complexity) and $\vec{x} = \{x_1, \ldots, x_n\}$ be a set of variables. Algebraic proof systems deal with polynomials in the polynomial ring over $\mathbb{F}$, i.e., multivariate polynomials in the variables $x_1, \ldots, x_n$ with coefficients from $\mathbb{F}$. A *monomial* m is a product of variables $m = \prod_{i=1}^{n} x_i^{e_i}$ for $e_i \in \mathbb{N}$. If $e_i \in \{0,1\}$ for all i we say that the monomial is *multilinear* (this is sometimes also referred to as being *square-free*). A *term* $t = \alpha m$ is a monomial m multiplied by some field element $\alpha \in \mathbb{F}$.[28]

7.5.1. Nullstellensatz

As our first example of an algebraic proof system we discuss *Nullstellensatz* introduced by Beame et al. [BIK+94]. A *Nullstellensatz refutation* of a set of polynomials $\mathcal{P} = \{p_i(\vec{x}) \mid i \in [m]\}$ in the polynomial ring $\mathbb{F}[\vec{x}]$ is a syntactic equality

$$\sum_{i=1}^{m} r_i(\vec{x}) \cdot p_i(\vec{x}) + \sum_{j=1}^{n} s_j(\vec{x}) \cdot (x_j^2 - x_j) = 1 \ , \tag{7.15}$$

where r_i, s_j are also polynomials in $\mathbb{F}[\vec{x}]$. In algebraic language, what this shows is that the multiplicative identity 1 of the field $\mathbb{F}$ lies in the polynomial ideal generated by $\mathcal{P} \cup \{x_j^2 - x_j \,|\, j \in [n]\}$. By Hilbert's Nullstellensatz, such a refutation (7.15) exists if and only if there is no common $\{0,1\}$-valued root for the set of polynomials $\mathcal{P}$.

[28] We remark that the exact meaning of the terms "monomial" and "term" varies in the literature, but our choice of terminology here seems well in line with common usage in proof complexity.

Nullstellensatz can also be viewed as a proof system for certifying the unsatisfiability of CNF formulas. In this setting we first translate clauses of the form

$$C = \bigvee_{x \in P} x \vee \bigvee_{y \in N} \overline{y} \tag{7.16a}$$

to polynomials

$$p(C) = \prod_{x \in P} (1 - x) \cdot \prod_{y \in N} y \ . \tag{7.16b}$$

As a concrete example, $D = x \vee y \vee \overline{z}$ gets translated to $p(D) = (1-x)(1-y)z = z - yz - xz + xyz$.[29] Then a Nullstellensatz refutation of $\{p(C_i) \mid i \in [m]\}$ can be viewed as a refutation of the CNF formula $F = \bigwedge_{i=1}^{m} C_i$. This is so since an assignment to the variables in F (where we think of 1 as true and 0 as false) is satisfying precisely if all the polynomials $\{p(C_i) \mid i \in [m]\}$ vanish, i.e., evaluate to 0, and a Nullstellensatz refutation rules out that such a satisfying assignment exists.

The *size* of a Nullstellensatz refutation is defined to be the total number of monomials encountered when all products of polynomials are expanded out as linear combinations of monomials. To be more precise, let $mSize(p)$ denote the number of monomials in a polynomial p written as a linear combination of monomials (so that for our example clause $D = x \vee y \vee \overline{z}$ above we have $mSize(p(D)) = 4$). Then the size of a Nullstellensatz refutation of the form (7.15) is

$$\sum_{i=1}^{m} mSize\big(r_i(\vec{x})\big) \cdot mSize\big(p_i(\vec{x})\big) + \sum_{j=1}^{n} 2 \cdot mSize\big(s_j(\vec{x})\big) \ . \tag{7.17}$$

We remark that this is not the only possible way of measuring size. It can be noted that the definition (7.17) is quite wasteful in that it forces us to represent the proof in a very inefficient way. Other papers in the so-called *semialgebraic* proof complexity literature, such as [GHP02b, KI06, DMR09], instead define size in terms of the polynomials in isolation, more along the lines of

$$\sum_{i=1}^{m} \big(mSize\big(r_i(\vec{x})\big) + mSize\big(p_i(\vec{x})\big)\big) + \sum_{j=1}^{n} \big(mSize\big(s_j(\vec{x})\big) + 2\big) \tag{7.18}$$

or as the bit size or "any reasonable size" of the representation of all polynomials $r_i(\vec{x}), p_i(\vec{x}), s_j(\vec{x})$. However, our definition (7.17) is consistent with the general definition of size for so-called algebraic and semialgebraic proof systems

[29] Note that in this translation we are thinking of 1 as *true* and 0 as *false*, which is standard in proof complexity. It can be argued, though, that in the specific context of algebraic proof systems a more natural convention is to flip the values and identify 0 with *true* and 1 with *false*, just as a clause evaluating to true is identified with the corresponding polynomial evaluating to 0. If we adopt this flipped convention, then $x \vee y \vee \overline{z}$ would be translated to $xy(1-z) = xy - xyz$. There is no clear consensus on this matter in the algebraic proof complexity literature, however, and so for simplicity we will identify 0 with *false* and 1 with *true* throughout this survey to be consistent with how other proof systems view the constants 0 and 1.

in [ALN16, Ber18, AO19], and, in particular, matches the size definition in the other algebraic proof systems that will be discussed later in this section.[30]

A much more well-studied measure for Nullstellensatz than size is *degree*, which we define as $\max\{\deg(r_i(\vec{x}) \cdot p_i(\vec{x})), \deg(s_j(\vec{x}) \cdot (x_j^2 - x_j))\}$. In order to prove a lower bound d on Nullstellensatz degree for refuting $\mathcal{P}$, one can construct a *d-design*, which is a map D from degree-d polynomials in $\mathbb{F}[\vec{x}]$ to $\mathbb{F}$ such that

1. D is linear, i.e., $D(\alpha p + \beta q) = \alpha D(p) + \beta D(q)$ for $\alpha, \beta \in \mathbb{F}$;
2. $D(1) = 1$;
3. $D(rp) = 0$ for all $p \in \mathcal{P}$ and $r \in \mathbb{F}[\vec{x}]$ such that $\deg(rp) \leq d$;
4. $D(x^2 s) = D(xs)$ for all $s \in \mathbb{F}[\vec{x}]$ such that $\deg(s) \leq d - 2$.

Designs provide a characterization of Nullstellensatz degree in that there is a d-design for $\mathcal{P}$ if and only if there is no Nullstellensatz refutation of $\mathcal{P}$ in degree d (this is clearly spelled out in [Bus98] but is mentioned there to have been known before). The only-if direction is clear — applying D to a purported Nullstellensatz refutation of the form (7.15) yields 0 on the left-hand side but 1 on the right-hand side, which is a contradiction. The if-direction requires more work, but follows from linear programming duality.

Lower bounds on Nullstellensatz degree have been proven for formulas encoding combinatorial principles such as the pigeonhole principle [BCE+98] and pebbling contradictions [BCIP02], which have already been described in previous sections, and also for other formulas encoding the ***induction principle*** [BP98b], ***house-sitting principle*** [CEI96, Bus98], and ***matching*** [BIK+97], most of which we will not discuss further in this survey.

It seems fair to say that research in algebraic proof complexity soon moved on from Nullstellensatz to stronger systems such as *polynomial calculus* [CEI96], which we will discuss shortly. Briefly (and somewhat informally), the difference between Nullstellensatz and polynomial calculus is that in the latter proof system the proof that 1 lies in the ideal generated by $\mathcal{P} \cup \left\{x_j^2 - x_j \,\middle|\, j \in [n]\right\}$ can be constructed dynamically by a step-by-step derivation, which sometimes makes it possible to decrease both degree and size significantly (while, in the opposite direction, the many lower bounds on degree and size later established for polynomial calculus, as discussed in Section 7.5.2, certainly apply also to Nullstellensatz). The Nullstellensatz proof system has seen somewhat of a revival in a recent line of works [RPRC16, PR17, PR18, GKRS19, dRMN+20] showing that Nullstellensatz degree lower bounds can be "lifted" to lower bounds in stronger computational models. We will briefly discuss lifting in Section 7.7.1 and then see some examples of these ideas can be used in proof complexity in the rest of Section 7.7. The size complexity measure for Nullstellensatz has also received attention in recent papers such as [Ber18, AO19, dRNMR19].

When proving lower bounds for algebraic proof systems it is often convenient

[30] We remark that in the end the difference between these two theoretical size measures is not too important, since the two measures (7.17) and (7.18) are at most a square apart, and when measuring size in proof complexity we typically focus on the distinction between polynomial and superpolynomial. In addition, and more importantly, when we are dealing with k-CNF formulas with $k = \mathrm{O}(1)$, as we are mostly doing in this survey, then the two size definitions are the same up to a constant factor 2^k. We refer the reader to Section 2.4 in [AH19] for a more detailed discussion of the definition of proof size in algebraic and semialgebraic proof systems.

to consider a *multilinear* setting where refutations are presented in the quotient ring $\mathbb{F}[\vec{x}]/\{x_j^2 - x_j \mid j \in [n]\}$. Since the Boolean axioms $x_j^2 - x_j$ are no longer needed, the refutation (7.15) can be written simply as

$$\sum_{i=1}^{m} r_i(\vec{x}) \cdot p_i(\vec{x}) = 1 \ , \tag{7.19}$$

where we assume that all results of multiplications are implicitly multilinearized. It is clear that any refutation on the form (7.15) remains valid after multilinearization, and so the size and degree measures can only decrease in a multilinear setting.

7.5.2. Polynomial Calculus

The proof system *polynomial calculus (PC)* was introduced in [CEI96] to model Gröbner basis computations (and was originally called the "Gröbner basis proof system," although by now the name "polynomial calculus" is firmly established). As in Nullstellensatz, the setup is that we have a set of polynomials $\mathcal{P}$ over variables $x_1, \ldots, x_n$, where the polynomial coefficients live in some field $\mathbb{F}$. The goal is to prove that the polynomials $\mathcal{P} \cup \{x_j^2 - x_j \,|\, j \in [n]\}$ have no common root. When operating with CNF formulas, we first translate the clauses to polynomials using the transformation from (7.16a) to (7.16b).

It is important to observe that, from an algebraic point of view, the variables can take as values any elements in the field $\mathbb{F}$. Hence, we need to add constraints enforcing 0/1 assignments. We are also allowed to take linear combinations of polynomials, or to multiply a polynomial by any monomial, since any common root for the original polynomials is preserved under such operations. This leads to the following set of derivation rules for polynomial calculus:

$$\textit{Boolean axioms} \quad \frac{}{x_j^2 - x_j} \tag{7.20a}$$

$$\textit{Linear combination} \quad \frac{p \qquad q}{\alpha p + \beta q} \ \ (\alpha, \beta \in \mathbb{F}) \tag{7.20b}$$

$$\textit{Multiplication} \quad \frac{p}{mp} \ \ (m \text{ any monomial}) \tag{7.20c}$$

A polynomial calculus refutation of $\mathcal{P}$ also allows polynomials from $\mathcal{P}$ as axioms; the refutation ends when 1 has been derived, showing that the polynomial equations have no common root, or equivalently when $\mathcal{P}$ is an translation of a CNF formula, that this formula is unsatisfiable. The polynomial calculus proof system is sound and complete, not only for CNF formulas but also for inconsistent systems of polynomial equations in general. As for Nullstellensatz, we can consider the setting where all polynomials are multilinear, because any higher powers of variables can always be eliminated using the Boolean axioms (7.20a).

To define the complexity measures of *size*, *degree*, and *space*, we write out the polynomials in a refutation as linear combinations of monomials. Then the *size* of a refutation, which is the analogue of resolution length, is the total number of

monomials in the refutation (counted with repetitions), the *degree*, corresponding to resolution width, is the largest degree of any monomial in it, and the *(monomial) space*,, which is the analogue of resolution (clause) space, is the maximal number of monomials in memory at any point during the refutation (again counted with repetitions). One can also define a *length* measure for polynomial calculus, which is the number of derivation steps, but this can be exponentially smaller than the size, which is the more relevant measure to study here.[31]

The representation of a clause $\bigvee_{i=1}^{n} x_i$ as a polynomial in polynomial calculus is $\prod_{i=1}^{n}(1-x_i)$, which means that the number of monomials is exponential in the clause width. This problem arises only for positive literals, however — a large clause with only negative literals is translated to a single monomial. To get a cleaner and more symmetric treatment, in [ABRW02] the proof system *polynomial calculus (with) resolution*, or *PCR* for short, was introduced. The idea is to add an extra set of parallel formal variables x'_j, $j \in [n]$, sometimes referred to as *twin variables*, so that positive and negative literals can both be represented in an efficient fashion.

Lines in a PCR proof are polynomials over the ring $\mathbb{F}[x_1, x'_1, \ldots, x_n, x'_n]$, where as before $\mathbb{F}$ is some field. We have all the axioms and rules of polynomial calculus plus the axiom

$$\textit{Negation} \quad \frac{}{x_j + x'_j - 1} \ . \tag{7.20d}$$

Size, length, and degree are defined as for polynomial calculus, and the (monomial) space of a PCR refutation is again the maximal number of monomials in any configuration counted with repetitions.

It is important to understand that from an algebraic point of view the variables x_j and x'_j are completely independent. The point of the negation rule, therefore, is to force x_j and x'_j to take opposite values in $\{0, 1\}$, so that they respect the intended meaning of negation. It is worth noting that in actual Gröbner basis calculations one would not have both x_j and x'_j, so the introduction of "twin variables" is just to get a nicer proof system from a theoretical point of view. Our example clause $D = x \vee y \vee \overline{z}$ is rendered as $x'y'z$ in PCR.

One gets the same degree bounds for polynomial calculus resolution as in polynomial calculus, (just substitute $1 - x$ for x'), but one can potentially avoid an exponential blow-up in size measured in the number of monomials (and thus also for space). There are k-CNF formulas for which PCR is exponentially more powerful than PC with respect to size [dRLM$^+$20]. In PCR, monomial space is a natural generalization of clause space since every clause translates into a monomial as just explained in the example above.

Clearly, PC and PCR are very closely related, and in what follows we will sometimes be a bit sloppy and write just "polynomial calculus" when the distinction between the two is not important. We write "polynomial calculus resolution" or "PCR" to highlight when a claim only holds for polynomial calculus with twin variables for positive and negative literals.

[31] In fact, if we consider the multilinear setting, where there are no Boolean axioms and instead multiplication is defined to return the multilinearized result, then it is not hard to show that any CNF formula with m clauses over n variables can be refuted in polynomial calculus in length $\mathrm{O}(mn)$. See, e.g., [MN15] for a proof of this fact.

$$\frac{x \vee \overline{y} \vee z \qquad \overline{y} \vee \overline{z}}{x \vee \overline{y}}$$

(a) Resolution step.

$$\dfrac{\boldsymbol{x'yz'} \qquad \dfrac{\dfrac{\boldsymbol{yz}}{x'yz} \qquad \dfrac{z' + z - 1}{x'yz' + x'yz - x'y}}{-x'yz' + x'y}}{\boldsymbol{x'y}}$$

(b) Corresponding PCR derivation.

Figure 7.13: Example of simulation of resolution step by PCR.

7.5.3. Nullstellensatz, Polynomial Calculus, and Resolution

Polynomial calculus resolution can simulate resolution efficiently with respect to length/size, width/degree, and space simultaneously simply by mimicking refutations step by step. This means that all worst-case upper bounds for resolution immediately carry over to PCR. For an example of how this works, see the simulation of the resolution step in Figure 7.13a by the derivation in Figure 7.13b, where the polynomials corresponding to the simulated clauses are in boldface.

Polynomial calculus can be strictly stronger than resolution with respect to size and degree. For instance, over GF(2) it is not hard to see that Tseitin formulas can be refuted in size $\mathrm{O}(N \log N)$ and degree $\mathrm{O}(1)$ by doing Gaussian elimination. Another example are the onto functional pigeonhole principle (FPHP) formulas (7.7a)–(7.7d), which were shown to be easy in [Rii93]. It remains open whether such separations can be found also for space, however.

Open Problem 7.5. Prove (or disprove) that polynomial calculus resolution is strictly stronger than resolution with respect to space.[32]

The proof systems Nullstellensatz and polynomial calculus without twin variables are *incomparable* to resolution with respect to size/length — there are formulas for which both Nullstellensatz and PC are exponentially more efficient than resolution, and other formulas for which resolution is exponentially better.

7.5.4. Size and Degree for Nullstellensatz and Polynomial Calculus

A lot of what is known about length versus width in resolution carries over to size versus degree in polynomial calculus, whereas Nullstellensatz is mostly different. It is not hard to show that for both Nullstellensatz and polynomial calculus upper bounds on degree imply upper bounds on size, in the sense that if a CNF formula over n variables can be refuted in degree d, then such a refutation can be carried out in size $n^{\mathrm{O}(d)}$. This is qualitatively similar to the bound for resolution, although the arguments are a bit more involved. Just as for resolution, this upper bound has been proven to be tight up to constant factors in the exponent for polynomial calculus [ALN16], and it follows from [LLMO09] that this also holds for Nullstellensatz.

[32] A constant-factor separation is claimed in [ABRW02], but this result is for the slightly different monomial space measure where every distinct monomial is only charged once but an arbitrary number of copies of this monomial (in different polynomials) can be had for free.

In the other direction, a lower bound on size in terms of degree exactly analogous to the size-width bound (7.9) for resolution [BW01] holds also for polynomial calculus, as shown in [IPS99]. For Nullstellensatz it is not possible to obtain lower bounds on size from degree lower bounds in this way, and pebbling formulas provide a counter-example [BCIP02].

Interestingly, the paper [IPS99] is a precursor to [BW01], and although it was far from obvious at the time it turns out that one can run exactly the same proof for both resolution and polynomial calculus. As for resolution, the ordering principle formulas in (7.11a)–(7.11d) witness the optimality of this size-degree lower bound, as shown by [GL10b]. As for resolution, almost all size lower bounds are derived via degree lower bounds.

The basic tool for proving polynomial calculus degree lower bounds is that of R-operators, which are analogues of the d-designs used for Nullstellensatz. As proposed in [Raz98], the idea is to give an overapproximation of what polynomials can be derived in degree at most d by defining an operator R on multilinear polynomials such that all degree-d consequences of the axioms are contained in the set $\{p \mid R(p) = 0\}$. The degree lower bound then follows by showing that $R(1) \neq 0$.

Formally, let $d \in \mathbb{N}^+$ be a positive integer. Suppose that there exists a linear operator R on a set $\mathcal{P}$ of (multilinear) polynomials of degree at most d with the following properties:

1. $R(1) \neq 0$.
2. $R(f) = 0$ for all axioms $f \in \mathcal{P}$.
3. For every term t with $Deg(t) < d$ and every variable x it holds that $R(xt) = R(xR(t))$.

Then any polynomial calculus refutation of $\mathcal{P}$ requires degree strictly greater than d. (Note that here we can restrict our attention to PC without twin variables for literals, since the degree measure is the same for PC and PCR.)

The proof of this claim is not hard. The basic idea is that R will map all axioms to 0 by property 2, and further derivation steps in degree at most d will yield polynomials that also map to 0 by the linearity of R and property 3 (where we use that without loss of generality we can implement multiplication by a monomial by multiplying by all variables in it one by one). But then property 1 implies that no derivation in degree at most d can reach 1 and establish contradiction. However, constructing such operators to obtain degree lower bounds seems much harder than proving resolution width lower bounds, and the technical machinery is much less well developed.

With the exception of Tseitin formulas and onto functional pigeonhole principle (FPHP) formulas, all the formulas discussed in detail in Section 7.4.1 are equally hard also with respect to polynomial calculus size, which can be shown via degree lower bounds arguments:

- Hardness of the standard CNF encoding (7.7a)–(7.7b) of pigeonhole principle formulas[33] follows from [AR03], with some earlier work on other non-

[33] Here a twist is needed since these formulas have high initial degree, but we will not go into this. The most elegant solution is to consider so-called graph PHP formulas as discussed in, e.g., [BW01, MN15].

CNF encodings in [Raz98, IPS99]. The proof in [AR03] works also if onto clauses (7.7d) are added, and more recently it was shown in [MN15] that FPHP formulas with clauses (7.7a)–(7.7c) are also hard (whereas with both onto and functionality axioms added the formulas are easy, as noted above).

- Strong degree and size lower bounds on random k-CNF formulas were shown in [BI99] for polynomial calculus over fields of characteristic distinct from 2, and lower bounds in any characteristic including 2 were established by different methods in [AR03].
- For the subset cardinality formulas in Figure 7.10, polynomial calculus degree and size lower bounds were obtained in [MN14].
- Also, "Tseitin formulas with the wrong modulus" are hard — one can define Tseitin-like formulas encoding counting modulo primes q, and such formulas are hard over fields of characteristic $p \neq q$ [BGIP01, AR03].

We also wish to discuss briefly two formula families for which we currently do not know how to prove lower bounds on polynomial calculus degree or size. Recall that a *legal k-colouring* of an undirected graph $G = (V, E)$ is a mapping $\chi : V \to [k]$ such that for every edge $(u, v) \in E$ it holds that $\chi(u) \neq \chi(v)$. The ***k-colouring formula*** for a graph G is the set of polynomials

$$\sum_{j=1}^{k} x_{v,j} - 1 \qquad v \in V(G), \tag{7.21a}$$

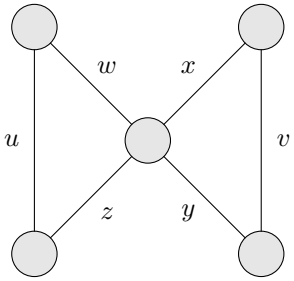
$$x_{v,j}x_{v,j'} \qquad v \in V(G),\ j \neq j' \in [k], \tag{7.21b}$$

$$x_{u,j}x_{v,j} \qquad (u, v) \in E(G),\ j \in [k], \tag{7.21c}$$

with the intended interpretation that $x_{v,j} = 1$ if vertex v has colour $\chi(v) = j$. It is clear that these polynomials have a common $\{0, 1\}$-valued root if and only if there exists a legal k-colouring, in which case we say that G is *k-colourable*. We wish to prove lower bounds on the complexity of disproving this formula for graphs G that are *not* k-colourable. (We remark that we can also represent (7.21a)–(7.21c) as a CNF formula by rewriting the only non-clausal constraints (7.21a) as the clause $\bigvee_{j=1}^{k} x_{v,j}$, but the exact representation does not matter for this discussion.)

For resolution, it was shown in [BCMM05] that if one samples a graph G at random with appropriate edge probabilities, then with overwhelming probability G is not k-colourable, but resolution still requires linear width, and hence exponential length, to refute the k-colouring formula over G.[34] This gives a very strong *average-case* lower bound over a whole distribution of inputs. For polynomial calculus, all that is known is that one can construct specific worst-case instances that are exponentially hard [LN17], but we cannot yet rule out that k-colouring would

[34] CNF versions (in the standard DIMACS format used by SAT solvers) of 3-colouring formulas over n-vertex random graphs with appropriate parameters can be obtained with the tool *CNFgen* [LENV17, CNF] using the command line `cnfgen kcolor 3 gnp` ⟨`n`⟩ ⟨`p`⟩ for $p = 5/(n-1)$. These formulas start getting quite challenging for SAT solvers around $n = 800$. (It should be noted that this edge density is slightly too low to get the theoretical guarantees from [AM99] that the graphs are not 3-colourable, and thus that the formulas are unsatisfiable, asymptotically almost surely, but empirically this seems to hold. Choosing $p = 5.5/(n-1)$ should be enough to also get theoretical guarantees, but yields more overconstrained formulas, meaning that n has to be chosen larger before the exponential hardness manifests itself.)



(a) Eulerian graph.

	$(u \vee w) \quad \wedge (w \vee x \vee y)$
	$\wedge\ (\overline{u} \vee \overline{w}) \quad \wedge (w \vee x \vee z)$
	$\wedge\ (u \vee z) \quad \wedge (w \vee y \vee z)$
$u + w = 1$	$\wedge\ (\overline{u} \vee \overline{z}) \quad \wedge (x \vee y \vee z)$
$u + z = 1$	$\wedge\ (v \vee x) \quad \wedge (\overline{w} \vee \overline{x} \vee \overline{y})$
$v + x = 1$	$\wedge\ (\overline{v} \vee \overline{x}) \quad \wedge (\overline{w} \vee \overline{x} \vee \overline{z})$
$v + y = 1$	$\wedge\ (v \vee y) \quad \wedge (\overline{w} \vee \overline{y} \vee \overline{z})$
$x + y + z + w = 2$	$\wedge\ (\overline{v} \vee \overline{y}) \quad \wedge (\overline{x} \vee \overline{y} \vee \overline{z})$

(b) Constraints. (c) CNF encoding.

Figure 7.14: Example of even colouring (EC) formula (satisfiable instance).

be an easy problem for Gröbner basis computations in an average-case setting (although it seems absurd that this would be the case).

Open Problem 7.6. Prove linear lower bound on polynomial calculus degree, and hence exponential lower bounds on size, for k-colouring formulas over randomly sampled graphs analogously to the average-case lower bound for resolution in [BCMM05].

Another quite intriguing family of benchmark formulas are the ***even colouring (EC) formulas*** constructed by Markström [Mar06] (see Figure 7.14 for an example). These formulas are defined over connected graphs with all vertices having even degree. For a fixed graph $G = (V, E)$ we consider every edge $e \in E$ to be a variable and let the formula consist of the (CNF encodings of) the constraints

$$\sum_{e \in E(v)} e = \deg(v)/2 \qquad\qquad v \in V, \tag{7.22}$$

where $E(v)$ denotes the set of edges incident to v. The constraints (7.22) require that the edges should be labelled 0/1 in such a way that for every vertex v in V the number of 0-edges and 1-edges incident to v is equal. If the total number of edges in the graph is even, then this formula is satisfiable — since the graph has all vertices of even degree it also has an Eulerian cycle, and we can fix any such cycle and label every second edge 0 and 1, respectively. If the number of edges is odd, however, then summing up (7.22) over all vertices yields $2 \cdot \sum_{e \in E(G)} e = |E(G)|$ which is clearly a contradiction since an odd number cannot be divisible by 2.

If the graph G is a good enough expander graph, then a fairly standard extension of the techniques in [BW01] can be used to prove that the even colouring formula over G requires linear width, and hence also exponential length, for resolution.[35] However, a moment of reflection reveals that over GF(2) it easy to

[35] To generate unsatisfiable even colouring formulas over 6-regular random graphs, which are expanding asymptotically almost surely, one can use *CNFgen* with the command line `cnfgen ec gnd ⟨n⟩ 6` for the number of vertices n in the graph chosen odd. For CDCL solvers, these formulas become noticeably hard around $n = 21$.

derive polynomials $\sum_{e \in E(v)} e - \deg(v)/2$ from (7.22), and summing these polynomials for all $v \in V$ yields $2 \cdot \sum_{e \in E(G)} e - |E(G)| = 1$ (where the calculations are in GF(2)). Thus, even colouring formulas are easy over GF(2), or more generally over any field of characteristic 2.

Open Problem 7.7. Are even colouring formulas over expander graphs hard for polynomial calculus over fields of characteristic distinct from 2?

7.5.5. Polynomial Calculus Space

We next turn to a discussion of space complexity. This measure does not make too much sense for Nullstellensatz, since refutations of the form (7.15) are presented in "one shot" and it is hard to talk about any intermediate results. Thus, our discussion of space will focus on polynomial calculus.

Recall that for resolution we measure space as the number of clauses in memory, and since clauses turn into monomials in polynomial calculus resolution the natural analogue here is monomial space (in our discussion of space we are always focusing on PCR, since this proof system allows for a more efficient representation of discjunctive clauses as discussed above). The first monomial space lower bounds were shown for pigeonhole principle formulas in [ABRW02]. These formulas have wide axioms, however, and if one applies the 3-CNF conversion from (7.1a) to (7.1b) the lower bound proof breaks down.

Monomial space lower bounds for formulas of bounded width were proven only in [FLN+15] for an obfuscated 4-CNF version of PHP formulas. This was followed by optimal, linear lower bounds for random 4-CNF formulas [BG15], and then for Tseitin formulas over expanders but with the added assumptions that either these graphs are sampled randomly or one adds two copies of every edge to get a multigraph [FLM+13].[36] Somewhat intriguingly, none of these papers could show any nontrivial lower bounds for any 3-CNF formulas. This barrier was finally overcome by [BBG+17], where optimal lower bounds for random 3-CNF formulas were established. However, the following open problems show that we still do not understand polynomial calculus space very well.

Open Problem 7.8. Prove linear lower bounds on polynomial calculus space for refutations of Tseitin formulas over d-regular expander graphs for $d = 3$ or even $d > 3$ using no other assumptions than expansion only.

Open Problem 7.9. Prove that pigeonhole principle formulas require large monomial space in polynomial calculus even when converted to 3-CNF.

Another intriguing question is whether an analogue of the fact that resolution width is a lower bound for resolution space (7.12) holds also for polynomial calculus.

Open Problem 7.10. Is it true that space $\geq$ degree $- \mathrm{O}(1)$ for k-CNF formulas in polynomial calculus?

[36] It is worth noting that these space lower bounds hold for any characteristic, so although Tseitin formulas have small-size refutations over GF(2), such refutations still require large space.

For a long time, essentially nothing was known about this problem, except that the work [FLM+13] made what can be described as some limited progress by showing that if a formula requires large resolution width (which is a necessary, but not sufficient, condition for high degree), then the XOR-substituted version of the formula (as in (7.13a)–(7.13c)) requires large monomial space. When applied to Tseitin-like formulas over expander graphs, this result yields an optimal separation of space and degree. Namely, it follows that these formulas can be refuted in degree $O(1)$ but require space $\Omega(N)$. To obtain such separations we have to commit to a finite characteristic p of the underlying field, however, and the formulas encoding counting mod p will separate space and degree only for fields of this characteristic. It would be nice to get a separation that would work in any characteristic, and the candidate formulas to obtain such a result readily present themselves.

Open Problem 7.11. Prove (or disprove) that XOR-substituted pebbling formulas as in Figure 7.12 require monomial space lower-bounded by the pebbling space of the underlying DAG (which if true would yield an essentially optimal space-degree separation independent of the field characteristic).

Recently, some quite exciting news regarding Open Problem 7.10 was announced [GKT19], namely that the PCR monomial space of refuting a formula is lower-bounded by the square root of the resolution refutation width (which, as mentioned above, is stronger than the corresponding lower bound in terms of degree, since resolution width can be much larger than polynomial calculus degree). This also implies lower bounds for the formulas mentioned in Open Problems 7.8 and 7.9 and provides a separation between degree and space that is independent of the characteristic as called for in Open Problem 7.11, although all of these results are off by a square root from what would have been expected. It is not clear whether the bounds obtained in this way are tight or not.

7.5.6. Trade-off Results for Nullstellensatz and Polynomial Calculus

When it comes to trade-offs in polynomial calculus we again recognize most of the picture from resolution, but there are also some differences. Here is a summary of what is known (where the upper bounds in all of the results hold for PC, i.e., polynomial calculus without twin variables, whereas the lower bound apply also for the PCR proof system with twin variables):

- For space versus degree in polynomial calculus we know strong, essentially optimal trade-offs from [BNT13], and the formulas exhibiting such trade-offs are the same vanilla pebbling contradictions as for resolution (for which we get exactly the same bounds as in Section 7.4.4). However, it is *not* known whether a result analogous to [BN20] holds, i.e., whether polynomial calculus refutations in small degree can require superlinear space.
- The paper [BNT13] also showed strong size-space trade-offs, and again the formulas used are substituted pebbling contradictions over appropriate DAGs and Tseitin formulas over long, narrow grids. Here there is some loss in parameters as compared to resolution, however, which seems to be due to limitations of the proof techniques rather than to actual differences

in formula behaviour, Also, the Tseitin formula trade-off results do not hold over fields of characteristic 2. Overall, though, the size-space trade-offs are very similar to the length-space trade-offs for resolution discussed in Section 7.4.4, and we omit the details.

- The size blow-up in [IPS99] when degree is decreased is necessary, and can be obtained for k-CNF formulas of constant width $k = \mathrm{O}(1)$, by using the same ideas as in [Tha16] and then adding a number of extra tweaks to make the argument go through in polynomial calculus as shown in [LNSS20].

In view of the above, we have two final open problems about polynomial calculus that we want to highlight in this section.

Open Problem 7.12. Establish size-space trade-offs for polynomial calculus that hold regardless of the characteristic of the underlying field (in contrast to [BNT13]).

Open Problem 7.13. Are there formulas over n variables for which polynomial calculus refutations in degree d require monomial space almost n^d, or at least $\omega(n)$?

We wish to mention also that for the weaker Nullstellensatz proof systems size-degree trade-offs were recently shown in [dRNMR19]. For instance, there is a family of 3-CNF formulas $\{F_n\}_{n=1}^{\infty}$ of size $\Theta(n)$ such that:

1. There is a Nullstellensatz refutation of F_n in degree $\mathrm{O}\big(\sqrt[6]{n}\log n\big)$.
2. There is a Nullstellensatz refutation of F_n of nearly linear size $\mathrm{O}(n^{1+\epsilon})$ and degree $\mathrm{O}\big(\sqrt[3]{n}\log n\big)$.
3. Any Nullstellensatz refutation of F_n in degree at most $\sqrt[3]{n}$ must have exponential size.

The formulas F_n are vanilla pebbling contradictions without substitution generated from suitably chosen graphs.

7.5.7. Algebraic SAT Solving

We conclude this section with a(n all too) brief discussion of algebraic SAT solving. There seems to have been quite some excitement, at least in the theory community, about the Gröbner basis approach to SAT solving after the paper [CEI96] appeared. However, the hoped for improvement in performance from this method failed to materialize in practice. Instead, the CDCL revolution happened...

Some Gröbner basis SAT solvers have been developed, the most notable example perhaps being *PolyBoRi* [BD09, BDG+09], but these solvers do not seem competitive with resolution-based solvers (and, sadly, *PolyBoRi* is no longer maintained). Some SAT solvers such as *March* [HvM06] and *CryptoMiniSat* [Cry] successfully implement Gaussian elimination [HvM05], but this is only very limited form of polynomial calculus reasoning.

Is it harder to build good algebraic SAT solvers than CDCL solvers? Or is it just that too little work has been done? (Witness that it took over 40 years for resolution-based SAT solvers to become really efficient.) Or is it perhaps a little bit of both? It can be noted in this context that it was recently shown

that efficient proof search is NP-hard for Nullstellensatz and polynomial calculus [dRGN+20] (strengthening an earlier paper [GL10a]) — i.e., these proof systems are not automatable — but it is not clear that this should be an obstacle in practice, since very successful SAT solvers have been built on top of resolution despite the non-automatability of this proof system [AM19].

Whatever the answer is to these questions, it seems clear that one needs to find some kind of shortcut in order to use Gröbner bases for efficient SAT solving. A full Gröbner basis computation does too much work, since it allows us not only to decide satisfiability but also to read off the number of satisfying assignments, which is widely believed to be a strictly harder problem.

This slightly downbeat discussion of algebraic SAT solving should not be taken to mean that algebraic methods cannot be used for successfully solving hard combinatorial problems, however. To give a positive example, in the sequence of papers [DLMM08, DLMO09, DLMM11], a body of work that was recognised with the *INFORMS Computing Society Prize 2010*, the authors solve graph colouring problems (with great success) essentially by constructing Nullstellensatz certificates of non-colourability. Hence, for some NP-complete problems it seems that even lowly Nullstellensatz can be a quite powerful approach.

Another very interesting line of work is exemplified by the papers [RBK17, RBK18, KBK19] using Gröbner bases computations to attack the challenging problem of verifying multiplier circuits. As a part of this work, the authors develop a formal proof logging system to certify correctness, called *practical algebraic calculus (PAC)*, and this proof system is nothing other than polynomial calculus (but with the field $\mathbb{F}$ chosen to be the rational numbers $\mathbb{Q}$ rather than a finite field).

7.6. Cutting Planes and Pseudo-Boolean Solving

The *cutting planes* proof system [CCT87], which formalizes the integer linear programming algorithm in [Gom63, Chv73] and underlies so-called *pseudo-Boolean (PB) solvers*, operates with linear inequalities over the reals with integer coefficients. To reason about CNF formulas in cutting planes, the disjunctive clauses are translated to linear inequalities, which are then manipulated to derive a contradiction. Thus, the question of Boolean satisfiability is reduced to the geometry of polytopes over the real numbers. Just as algebraic proof systems can deal not only with translated CNF formulas but with arbitrary sets of polynomials, cutting planes can operate on arbitrary 0-1 integer linear constraints, which we will also refer to as *pseudo-Boolean constraints*. A *pseudo-Boolean formula* is a set of pseudo-Boolean constraints (also known as a 0-1 *integer linear program*).

In the standard proof complexity setting, we use only positive literals (unnegated variables) and identify $\overline{z}$ with $1-z$ so that, for instance, the clause $x \vee y \vee \overline{z}$ gets translated to $x + y + (1 - z) \geq 1$, or $x + y - z \geq 0$ after we have moved all integer constants to the right-hand side. However, in order to give a description of cutting planes that is helpful also when we want to reason about pseudo-Boolean solvers, and in order to get compact notation, it is more useful to keep negated literals as variables in their own right, and to insist that all inequalities consist of linear combinations of (positive or negative) literals with only non-negative

coefficients.[37] It will also be convenient here to use the notation x^σ, $\sigma \in \{0,1\}$, mentioned in Section 7.2, where we recall that $x^1 = x$ and $x^0 = \overline{x}$. With this notation, assuming that our set of variables is $\{x_1, \ldots, x_n\}$ we can write all linear constraints in *normalized form*

$$\sum_{i\in[n],\, \sigma\in\{0,1\}} a_i^\sigma x_i^\sigma \geq A \ , \tag{7.23}$$

where for all $i \in [n]$ and $\sigma \in \{0,1\}$ it holds that $a_i^\sigma \in \mathbb{N}$ and $\min\{a_i^0, a_i^1\} = 0$ (the latter condition specifies that variables occur only with one sign in any given inequality), and where $A \in \mathbb{N}^+$ (this constant term is often referred to as the *degree of falsity*, or just *degree*, in an applied pseudo-Boolean solving context). In what follows, all expressions of the type (7.23) are supposed to be in normalized form, and all sums are assumed to be taken over all combinations of $i \in [n]$ and $\sigma \in \{0,1\}$ except as specified under the summation sign.

If the input is a CNF formula F we just view every clause $C \in F$ of the form

$$C = x_1^{\sigma_1} \vee x_2^{\sigma_2} \vee \cdots \vee x_w^{\sigma_w} \tag{7.24a}$$

as a linear constraint

$$x_1^{\sigma_1} + x_2^{\sigma_2} + \cdots + x_w^{\sigma_w} \geq 1 \ . \tag{7.24b}$$

That is, a disjunctive clause is simply a constraint on the form (7.23) where $a_i^\sigma \in \{0,1\}$ and $A = 1$ (in particular, our example clause $x \vee y \vee \overline{z}$ now becomes $x^1 + y^1 + z^0 \geq 1$). Hence, CNF formulas can be viewed as a special case of pseudo-Boolean formulas).

Pseudo-Boolean constraints can be exponentially more concise than CNF, as is shown by a comparison of the constraint

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \geq 3 \tag{7.25a}$$

with

$$\begin{aligned}
&(x_1 \vee x_2 \vee x_3 \vee x_4) \wedge (x_1 \vee x_2 \vee x_3 \vee x_5) \wedge (x_1 \vee x_2 \vee x_3 \vee x_6) \\
&\wedge (x_1 \vee x_2 \vee x_4 \vee x_5) \wedge (x_1 \vee x_2 \vee x_4 \vee x_6) \wedge (x_1 \vee x_2 \vee x_5 \vee x_6) \\
&\wedge (x_1 \vee x_3 \vee x_4 \vee x_5) \wedge (x_1 \vee x_3 \vee x_4 \vee x_6) \wedge (x_1 \vee x_3 \vee x_5 \vee x_6) \\
&\wedge (x_1 \vee x_4 \vee x_5 \vee x_6) \wedge (x_2 \vee x_3 \vee x_4 \vee x_5) \wedge (x_2 \vee x_3 \vee x_4 \vee x_6) \\
&\wedge (x_2 \vee x_3 \vee x_5 \vee x_6) \wedge (x_2 \vee x_4 \vee x_5 \vee x_6) \wedge (x_3 \vee x_4 \vee x_5 \vee x_6)
\end{aligned} \tag{7.25b}$$

(note that we are assuming here that it is not allowed to introduce new variables to encode the constraints — as discussed in Section 7.8, such extension variables can change the properties of proof systems dramatically). Constraints of the form

$$x_1^{\sigma_1} + x_2^{\sigma_2} + \cdots + x_w^{\sigma_w} \geq A \ , \tag{7.26}$$

i.e., such that $a_i^\sigma \in \{0,1\}$ holds for all coefficients a_i^σ, are called *cardinality constraints*, since they encode that at least A of the literals in the constraint are true. We can also have general pseudo-Boolean constraints such as, say, $x_1 + 2\overline{x}_2 + 3x_3 + 4\overline{x}_4 + 5x_5 \geq 7$.

[37] This is analogous to how twin variables were introduced for polynomial calculus in Section 7.5.2, except that for cutting planes this does not affect the reasoning power of the proof system in any significant way.

7.6.1. Pseudo-Boolean Rules of Reasoning

Not only are pseudo-Boolean constraints much more concise than clauses, but the rules used to manipulate them are also more powerful. Using the normalized form (7.23), the derivation rules in the standard proof complexity definition of cutting planes are

$$\textit{Literal axioms}\ \frac{}{x_i^\sigma \geq 0} \tag{7.27a}$$

$$\textit{Multiplication}\ \frac{\sum a_i^\sigma x_i^\sigma \geq A}{\sum c a_i^\sigma x_i^\sigma \geq cA}\ c \in \mathbb{N}^+ \tag{7.27b}$$

$$\textit{Addition}\ \frac{\sum a_i^\sigma x_i^\sigma \geq A \qquad \sum b_i^\sigma x_i^\sigma \geq B}{\sum (a_i^\sigma + b_i^\sigma) x_i^\sigma \geq A + B} \tag{7.27c}$$

$$\textit{Division}\ \frac{\sum c a_i^\sigma x_i^\sigma \geq A}{\sum a_i^\sigma x_i^\sigma \geq \lceil A/c \rceil}\ c \in \mathbb{N}^+ \tag{7.27d}$$

where in the addition rule (7.27c) we implicitly assume that the result is rewritten in normalized form. Let us illustrate this by a small example. It is important to note that when we add literals of opposite sign, the result is

$$x_i^1 + x_i^0 = x_i^1 + \left(1 - x_i^1\right) = 1 \tag{7.28}$$

(which is just another way of saying that it will always be the case that exactly one of the literals x_i and $\overline{x}_i$ is true). If we have the two constraints

$$x + 2y + 3z + 4\overline{w} \geq 5 \tag{7.29}$$

and

$$3x + 2\overline{y} + \overline{z} + \overline{w} \geq 3\ , \tag{7.30}$$

then by applying the addition rule (7.27c) we get the expression

$$(1+3)x + (2-2)y + (3-1)z + (4+1)\overline{w} \geq 5 + 3 - (2+1) \tag{7.31a}$$

which in the normalized form (7.23) becomes

$$4x + 2z + 5\overline{w} \geq 5 \tag{7.31b}$$

(where we suppress terms with zero coefficients). We note that when adding (7.29) and (7.30) to obtain (7.31b) the coefficients for y and $\overline{y}$ cancel so that this variable disappears. In general, when we add two constraints $\sum a_i^\sigma x_i^\sigma \geq A$ and $\sum b_i^\sigma x_i^\sigma \geq B$ such that there is a variable x_i and a $\sigma \in \{0,1\}$ for which $a_i^\sigma = b_i^{1-\sigma} > 0$, we say that this is an instance of *cancelling addition*.

More generally, when two linear constraints $\sum a_i^\sigma x_i^\sigma \geq A$ and $\sum b_i^\sigma x_i^\sigma \geq B$ share a variable x_j for which $a_j^\sigma > 0$ and $b_j^{1-\sigma} > 0$ hold, then we can multiply the constraints by the smallest numbers c_A and c_B such that $c_A a_j^\sigma = c_B b_j^{1-\sigma}$, and then apply cancelling addition. If we return to the constraints (7.29) and (7.30)

but now focus on the variable z, then we can multiply the second constraint by 3 and then add to get

$$\frac{x + 2y + 3z + 4\overline{w} \geq 5 \qquad \dfrac{3x + 2\overline{y} + \overline{z} + \overline{w} \geq 3}{9x + 6\overline{y} + 3\overline{z} + 3\overline{w} \geq 9}}{10x + 4\overline{y} + 7\overline{w} \geq 9} \tag{7.32}$$

More abstractly, writing $d = \gcd(a_j, b_j)$ for the greatest common divisor of a_j and b_j, so that the smallest numbers c_A and c_B such that $c_A a_j = c_B b_j$ are $c_A = b_j/d$ and $c_B = a_j/d$, the formal specification of this rule is

$$\frac{a_j x_j^\sigma + \sum_{i \neq j, \sigma} a_i^\sigma x_i^\sigma \geq A \qquad b_j x_j^{1-\sigma} + \sum_{i \neq j, \sigma} b_i^\sigma x_i^\sigma \geq B}{\sum_{i \neq j, \sigma} \big((b_j a_i^\sigma / d) + (a_j b_i^\sigma / d)\big) x_i^\sigma \geq b_j A/d + a_j B/d - a_j b_j / d} \tag{7.33}$$

(where as before we implicitly assume that the result of the linear combination is put into normalized form). Note that this can be viewed as a kind of generalization of the resolution rule (7.2) from disjunctive clauses to general linear constraints. We therefore refer to (7.33) as *generalized resolution* (or sometimes *cancelling linear combination*), and we say that the two constraints are *resolved over* x_j. This rule essentially goes back to Hooker [Hoo88, Hoo92] (although Hooker's definition is slightly different in that the cancelling addition has to be followed by a division step). It is worth noting, though, that one difference compared to the resolution rule for disjunctive clauses is that it may be possible to resolve the same pair of constraints over several different variables, which can yield different results (as in (7.31b) and (7.32) above).

Given a set of linear inequalities, one can show that there is no $\{0,1\}$-valued solution by using the cutting planes rules to derive the inequality $0 \geq 1$ from the given linear inequalities. It is clear that such a refutation can exist only if the instance is indeed unsatisfiable. The other direction also holds, but requires more work to establish [Chv73, CCT87]; for the special case of translations of CNF formulas, this follows from the cutting planes can simulate resolution as discussed in Section 7.7.2 below).

We want to highlight that in the division rule (7.27d) (which is also referred to as the *Chvátal-Gomory cut* rule) we can divide with the common factor c on the left and then *divide and round up* the constant term on the right to the closest integer, since we know that we are only interested in $\{0,1\}$-valued solutions. This division rule is where the power of cutting planes lies. (And, indeed, this is how it must be, since a moment of thought reveals that the other rules are sound also for real-valued variables, and so without the division rule we would not be able to distinguish sets of linear inequalities that have real-valued solutions but no $\{0,1\}$-valued solutions.)

It is not hard to see that we can modify the definitions slightly to obtain a more cleanly stated *general division* rule

$$\frac{\sum a_i^\sigma x_i^\sigma \geq A}{\sum \lceil a_i^\sigma / c \rceil x_i^\sigma \geq \lceil A/c \rceil} \; c \in \mathbb{N}^+ \tag{7.34}$$

without changing anything essential, since this rule can easily be simulated by using rules (7.27a) and (7.27d). Therefore, although the standard definition of division in the proof complexity literature is as in (7.27d), without loss of generality

we will use rule (7.34) (though it should be noted that the correctness of (7.34) hinges on the fact that we are using normalized form).

A small example just to illustrate how the rules can be combined is the derivation

$$\dfrac{\dfrac{6x+2y+3z\geq 5 \qquad \dfrac{x+2y+w\geq 1}{2x+4y+2w\geq 2}\ \textsf{Multiplication by 2}}{8x+6y+3z+2w\geq 7}\ \textsf{Addition}}{3x+2y+z+w\geq 3}\ \textsf{Division by 3} \tag{7.35}$$

where we note that in this case we do not lose any information in the final division step. This is not always true when using the general division rule (7.34) — for instance, a further division of $3x+2y+z+w\geq 3$ by 3 would yield the clause $x+y+z+w\geq 1$, which is a strictly weaker constraint.

Pseudo-Boolean solvers such as *Sat4j* [LP10] do not implement the full set of cutting planes derivation rules as presented above, however. To describe how they work, we need to introduce two other rules, namely

$$\textit{Weakening}\ \frac{\sum_{(i,\sigma)} a_i^\sigma x_i^\sigma \geq A}{\sum_{(i,\sigma),\ i\neq j} a_i^\sigma x_i^\sigma \geq A - \left(a_j^0 + a_j^1\right)} \tag{7.36a}$$

and

$$\textit{Saturation}\ \frac{\sum_{(i,\sigma)} a_i^\sigma x_i^\sigma \geq A}{\sum_{(i,\sigma)} \min\left\{a_i^\sigma, A\right\}\cdot x_i^\sigma \geq A}\ . \tag{7.36b}$$

The weakening rule is merely a convenient shorthand for one application each of the rules (7.27a), (7.27b), and (7.27c). Just as division, saturation is a special rule in that it is sound only for integral solutions. A second small toy example

$$\dfrac{\dfrac{2x+y+z+\geq 2 \qquad \dfrac{3\overline{x}+2y+2\overline{z}+u+w\geq 5}{3\overline{x}+2y+u+w\geq 3}\ \textsf{Weakening on } z}{7y+3z+2u+2w\geq 6}\ \textsf{Resolution on } x}{6y+3z+2u+2w\geq 6}\ \textsf{Saturation} \tag{7.37}$$

shows how weakening, generalized resolution, and saturation can be combined. (Again, the statement of the saturation rule in (7.36b) is crucially using normalized form.)

In the proof complexity literature the focus has been on investigating the general cutting planes proof system with the derivation rules (7.27a)–(7.27c) and (7.34). To understand the reasoning power of pseudo-Boolean solvers, however, it makes sense to study also other versions of the cutting planes proof system where (i) the saturation rule (7.36b) is used instead of the division rule (*cutting planes with saturation*), or (ii) all applications of (7.27b) and (7.27c) have to be combined into applications of the generalized resolution rule (7.33) (*cutting planes with resolution*), or (iii) both of these modifications are applied at the same time (*cutting planes with saturation and resolution*). We will return to these different versions of the cutting planes proof system in Section 7.7.4.

7.6.2. Conflict-Driven Pseudo-Boolean Solving

To explain why the different flavours of cutting planes just introduced are interesting from an applied point of view, we next turn to pseudo-Boolean solvers and how they can be used to determine the satisfiability of sets of pseudo-Boolean constraints, which we recall are referred to as pseudo-Boolean formulas.

One approach to solving pseudo-Boolean formulas is to convert them to CNF, either lazily by learning clauses from PB constraints during conflict analysis, as in *clasp* [GKKS09] and one of the version in the *Sat4j* library [LP10], or eagerly by re-encoding the whole formula to CNF (typically introducing new auxiliary variables, or extension variables) and running a CDCL solver as in, e.g., *MiniSat+* [ES06], *Open-WBO* [MML14], or *NaPS* [SN15]. In the context of cutting planes we are more more interested in solvers doing native pseudo-Boolean reasoning, such as *PRS* [DG02], *Galena* [CK05], *Pueblo* [SS06], *Sat4j* [LP10], and *RoundingSat* [EN18], so this is where our focus will be below (we mention, though, that related, but slightly different, ideas were also explored in *bsolo* [MM06]). Our discussion of pseudo-Boolean solvers cannot come anywhere close to doing full justice to the topic of pseudo-Boolean solving or the even richer area of pseudo-Boolean optimization — for this, we refer the reader to Chapter 7.11 in this handbook. Another source of much useful information is Dixon's PhD thesis [Dix04].

In our discussion of pseudo-Boolean solving, the standard setting is that the input is a PB formula without 0-1 solutions, and the goal of the solver is to decide that the formula is contradictory. For readers more interested in optimization, note that the above scenario arises also when the optimal solution has been found and the solver should prove that the (linear) objective function cannot be better than in the current solution.

Just as CDCL solvers can be viewed as searching for resolution proofs, we will see that the pseudo-Boolean solving techniques we will discuss generate proofs in different subsystems of cutting planes. Simplifying drastically, when building a pseudo-Boolean solver on top of cutting planes we have the following choices:

1. Boolean rule: (a) saturation or (b) division.
2. Linear combinations: (a) generalized resolution or (b) no restrictions.

As we will soon see, the choice of generalized resolution seems inherent in a conflict-driven setting, which is what we are focusing on here. However, which Boolean rule to prefer is less clear. Saturation was used in the seminal paper [CK05] and has also been the rule of choice in what is arguably the most popular pseudo-Boolean solver to date, namely *Sat4j* [LP10]. Division appeared only recently in *RoundingSat* [EN18] (although it was suggested in a more general integer linear programming setting in [JdM13]). We will return to a discussion of saturation versus division later, but let us first describe the general setup.

Naively, when generalizing CDCL to a pseudo-Boolean setting we just want to build a solver that decides on variable values and propagates forced values until conflict, at which point a new linear constraint is learned and the solver backtracks. To decide when constraints are propagating or conflicting it is convenient to use the concept of *slack*, which we define next.

ρ	$slack(C;\rho)$	Comment
$\emptyset$	8	Sum of coefficients minus degree of falsity.
$(\overline{x}_5)$	3	Propagates $\overline{x}_4$, since coefficient > slack.
$(\overline{x}_5, \overline{x}_4)$	3	Propagation does not change slack.
$(\overline{x}_5, \overline{x}_4, \overline{x}_3, x_2)$	-2	Conflict, since slack < 0.

Figure 7.15: Slack of $C \doteq x_1 + 2\overline{x}_2 + 3x_3 + 4\overline{x}_4 + 5x_5 \geq 7$ for different trails ρ.

The *slack* of a constraint $\sum_{i\in[n],\,\sigma\in\{0,1\}} a_i^\sigma x_i^\sigma \geq A$ under a partial assignment ρ (which we should think of as the variables currently assigned on the trail) measures how far $\sum_{i\in[n],\,\sigma\in\{0,1\}} a_i^\sigma x_i^\sigma \geq A$ is from being falsified by ρ, and is defined as

$$\operatorname{slack}\left(\sum_{i\in[n],\,\sigma\in\{0,1\}} a_i^\sigma x_i^\sigma \geq A; \rho\right) = \sum_{\rho(x_i^\sigma)\neq 0} a_i^\sigma - A \ , \tag{7.38}$$

i.e., the sum of the coefficients of all literals that are *not* falsified by ρ minus the constant term. To illustrate with a special case, the slack of a disjunctive clause is the number of non-falsified literals it the clause minus 1. The constraint $\sum_{i\in[n],\,\sigma\in\{0,1\}} a_i^\sigma x_i^\sigma \geq A$ is *conflicting* under ρ if

$$\operatorname{slack}\left(\sum_{i\in[n],\,\sigma\in\{0,1\}} a_i^\sigma x_i^\sigma \geq A; \rho\right) < 0 \tag{7.39}$$

and, for x_{i^*} not in the domain of ρ, it *propagates* $x_{i^*}^{\sigma^*}$ under ρ if

$$0 \leq \operatorname{slack}\left(\sum_{i\in[n],\,\sigma\in\{0,1\}} a_i^\sigma x_i^\sigma \geq A; \rho\right) < a_{i^*}^{\sigma^*} \tag{7.40}$$

(which is just another way of saying that we have to set $x_{i^*}^{\sigma^*}$ to true, or else there will be no way to satisfy the constraint). The above definitions might be easier to digest by studying the example in Figure 7.15 of how the slack changes for the constraint $C \doteq x_1+2\overline{x}_2+3x_3+4\overline{x}_4+5x_5 \geq 7$ under different partial assignments ρ on the trail. The initial slack is just the sum of the coefficients minus the degree. If x_5 is assigned to false, then x_4 is propagated to false according to (7.40). Assigning a variable to its propagated value does not change the slack. If now the solver for some other reason sets x_3 to false and x_2 to true, we have a conflict according to (7.39). Note that in contrast to clauses in CDCL, a pseudo-Boolean constraint can be conflicting even though not all variables in it have been assigned.

7.6.2.1. A First (Failed) Attempt at Pseudo-Boolean Conflict Analysis

Now we can give a slightly more detailed (though still incomplete) sketch of how we would like our pseudo-Boolean solver to work:

1. While there is no conflict, iteratively propagate all literals $x_{i^*}^{\sigma^*}$ such that there is a constraint for which $0 \leq \operatorname{slack}\big(\sum_{i\in[n],\,\sigma\in\{0,1\}} a_i^\sigma x_i^\sigma \geq A; \rho\big) < a_{i^*}^{\sigma^*}$ (i.e., add the literals to the current assignment ρ).

2. If there is no conflict, find some unassigned literal x_i^σ, decide to set it to true (i.e., add x_i^σ to ρ), and go to step 1.
3. Otherwise, fix some conflicting constraint $\sum_{i\in[n],\,\sigma\in\{0,1\}} a_i^\sigma x_i^\sigma \geq A$, i.e., such that $\mathrm{slack}\big(\sum_{i\in[n],\,\sigma\in\{0,1\}} a_i^\sigma x_i^\sigma \geq A; \rho\big) < 0$, and resolve it with the propagating constraints found in step 1 in reverse chronological order until we derive a constraint that is the analogue of an asserting clause.
4. Add this constraint to the formula, backtrack to the first point in the trail where the constraint is not falsified, and go to step 1.

From this description it is possible to see why the generalized resolution enters the picture in a natural way — the only time we use linear combinations is in step 3, where there are two constraints that have opposing opinions about what value the current variable under consideration should take.

The simple approach outlined above will not quite work. In what remains of this subsection, we first show an example why it fails, and then discuss what can be done to fix the problem. From the examples in Section 7.3 we can see that an important invariant during CDCL conflict analysis is that the assignment that is "left on the trail" always falsifies the currently derived clause. This means that every derived constraint "explains" the conflict by showing what assignments on the trail are inconsistent, and we can continue the conflict analysis until the derived constraint looks "nice," at which point the solver learns it and backtracks. The standard concept of "niceness" in CDCL is that the constraint should be *asserting*, i.e., that if we remove further literals from the trail in reverse chronological order until the first time when the learned constraint is not falsified, then at this point the constraint propagates some variable that flips an earlier assignment. When we generalized conflict-driven solving to a pseudo-Boolean setting, we would like the conflict analysis to work in the same way.

As a running example, let us consider the pseudo-Boolean formula consisting of the two constraints

$$C_1 \doteq 2x_1 + 2x_2 + 2x_3 + x_4 \geq 4 \tag{7.41a}$$

$$C_2 \doteq 2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 3 \tag{7.41b}$$

(which is just an obfuscated way of writing that a majority of the variables $\{x_1, x_2, x_3\}$ have to be true and false at the same time, but it is a simple enough example that we can use it to illustrate our main points). Note that both constraints have slack 3 under the empty assignment, which is larger than all coefficients, so there are no propagations at decision level 0. Suppose that the solver sets x_1 to false to get the trail

$$\rho_1 = \big(x_1 \xleftarrow{\mathsf{DEC}} 0\big) \ . \tag{7.42a}$$

Now $\mathit{slack}(C_1; \rho_1) = 1$, which is less than the coefficient 2 of x_2, so C_1 propagates x_2 to true, yielding

$$\rho_2 = \big(x_1 \xleftarrow{\mathsf{DEC}} 0, x_2 \xleftarrow{C_1} 1\big) \ . \tag{7.42b}$$

For the same reason, x_3 is also propagated to true by C_1 (note that, in contrast to CDCL, the same constraint can propagate several times). For this trail

$$\rho_3 = \big(x_1 \xleftarrow{\mathsf{DEC}} 0, x_2 \xleftarrow{C_1} 1, x_3 \xleftarrow{C_1} 1\big) \tag{7.42c}$$

the slack of the other constraint C_2 is $\mathit{slack}(C_2; \rho_3) = -1$, and we have reached a conflict.

Inspired by CDCL conflict analysis, we take the reason constraint $C_1 \doteq \mathsf{reason}(x_3, \rho_2)$ propagating the last literal x_3 under ρ_2 and resolve it over x_3 with the conflicting constraint C_2, i.e., performing the derivation step

$$\frac{2x_1 + 2x_2 + 2x_3 + x_4 \geq 4 \qquad 2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 3}{x_4 \geq 1}\ , \tag{7.43}$$

to obtain the resolvent $x_4 \geq 1 \doteq \mathsf{resolve}(C_1, C_2, x_3)$. But now we have a problem: the slack of the resolvent with respect to what remains on the trail is $\mathit{slack}(x_4 \geq 1; \rho_2) = 0$. This is no longer negative, so we have lost the invariant from CDCL that the constraint derived on the conflict side should be conflicting!

Taking a step back to analyse what happened, the reason for this failure is that it is in fact possible to satisfy both constraints C_1 and C_2 by extending ρ_2 with the assignment $x_3 = \frac{1}{2}$. Of course, this is not a Boolean assignment. However, taking linear combinations is not a Boolean rule but is sound also over the reals. For this reason, there is no way we can guarantee that the invariant of a conflicting constraint on the conflict side can be maintained if we use only the generalized resolution rule (7.33). Thus, we need to get some Boolean derivation rule into play.

7.6.2.2. Pseudo-Boolean Conflict Analysis Using Saturation

We will now describe how Chai and Kuehlmann [CK05] adapt conflict analysis to a pseudo-Boolean setting using the saturation rule. Saturation in itself cannot help fix our problem, because both constraints resolved in (7.43) are already saturated, as is the resolvent. But if we combine saturation with weakening of the reason constraint, then (perhaps somewhat counter-intuitively) we can get the conflict analysis to work. When resolving a propagating constraint C_{reason} on the reason side with the currently derived constraint C_{confl} on the conflict side (starting with the violated constraint and working our way back along the trail in reverse chronological order, as described above), we will iterate the following procedure:

1. weaken the current reason constraint C_{reason} on some non-falsified literal ℓ' (other than the literal ℓ propagated by C_{reason}, on which we want to apply resolution) to get $\mathsf{weaken}(C_{\text{reason}}, \ell')$;
2. saturate the weakened constraint to obtain $\mathsf{saturate}(\mathsf{weaken}(C_{\text{reason}}, \ell'))$;
3. compute the resolvent $\mathsf{resolve}(C_{\text{confl}}, \mathsf{saturate}(\mathsf{weaken}(C_{\text{reason}}, \ell')), \ell)$ of the weakened reason constraint with the conflicting constraint C_{confl} over the propagated literal ℓ;

until obtaining a resolved constraint that is conflicting. After each step in this iteration C_{reason} is updated to to $\mathsf{weaken}(C_{\text{reason}}, \ell')$, and when we finally get a conflicting resolvent we update C_{confl} to be this new constraint.

Let us first show how this works for our example, and then discuss why this is a correct approach in general. If we weaken $\mathsf{reason}(x_3, \rho_2) \doteq C_1$ on x_2, which is the first non-falsified literal that is not the one currently propagated, then we

```
while slack(resolve(C_confl, C_reason, ℓ); ρ) ≥ 0 do
    ℓ' ← literal in C_reason \ {ℓ} not falsified by ρ;
    C_reason ← saturate(weaken(C_reason, ℓ'));
end
return C_reason;
```

Figure 7.16: Chai-Kuehlmann reason reduction $\mathsf{reduceSat}(C_{\mathrm{confl}}, C_{\mathrm{reason}}, \ell, \rho)$.

get the following derivation:

$$\dfrac{\dfrac{\dfrac{2x_1 + 2x_2 + 2x_3 + x_4 \geq 4}{2x_1 + 2x_3 + x_4 \geq 2}\,\mathsf{Weakening\ on}\ x_2}{2x_1 + 2x_3 + x_4 \geq 2}\,\mathsf{Saturation} \qquad 2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 3}{2\overline{x}_2 + x_4 \geq 1}\,\mathsf{Resolution\ on}\ x_3 \tag{7.44}$$

Unfortunately, this does not solve the problem, since $2\overline{x}_2 + x_4 \geq 1$ again has slack 0 with respect to the trail ρ_2 in (7.42b).

We cannot weaken away x_3, since this is the propagating literal we want to resolve over, but we can weaken C_1 also on x_4, which is not falsified. This yields

$$\dfrac{\dfrac{\dfrac{\dfrac{2x_1 + 2x_2 + 2x_3 + x_4 \geq 4}{2x_1 + 2x_3 + x_4 \geq 2}\,\mathsf{Weakening\ on}\ x_2}{2x_1 + 2x_3 \geq 1}\,\mathsf{Weakening\ on}\ x_4}{x_1 + x_3 \geq 1}\,\mathsf{Saturation} \qquad 2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 3}{2\overline{x}_2 \geq 1}\,\mathsf{Resolution\ on}\ x_3 \tag{7.45}$$

and now we have $\mathit{slack}(2\overline{x}_2 \geq 1; \rho_2) = -1 < 0$, i.e., we have derived a new constraint that maintains the invariant of having negative slack with respect to what remains on the trail. This does not change if we saturate this constraint to get $\overline{x}_2 \geq 1$.

Although we have not formally defined anything like 1UIP pseudo-Boolean constraints — and, indeed, doing so requires some care — it should be clear that $\overline{x}_2 \geq 1$ is asserting. If we undo all decisions on the trail, then at top level we have $\mathit{slack}(\overline{x}_2 \geq 1; \emptyset) = 0$, so x_2 propagates to false yielding $\rho_4 = \left(x_2 \overset{\overline{x}_2 \geq 1}{\longleftarrow} 0\right)$. Looking at $C_1 \doteq 2x_1 + 2x_2 + 2x_3 + x_4 \geq 4$, we have $\mathit{slack}(C_1; \rho_4) = 1$, and since this is smaller than the coefficient 2 of x_1 and x_3 both variables propagate to true. This causes a conflict with $C_2 \doteq 2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 3$, and since no decisions have been made the solver can terminate and report that the formula consisting of the constraints (7.41a) and (7.41b) is indeed unsatisfiable.

The key to the pseudo-Boolean conflict analysis just described is that we apply a *reduction algorithm* on the reason constraint, combining weakening and saturation, to ensure that when the reduced reason constraint is resolved with the currently derived conflict constraint, then the result will be a new conflicting constraint. The pseudo-code for this reduction algorithm from [CK05] is given in Figure 7.16. But how do we know that this procedure is guaranteed to work?

Briefly, the reason is that slack is *subadditive*, i.e., if we take a linear combi-

```
while C_confl not asserting do
    ℓ ← literal assigned last on trail ρ;
    if ¬ℓ occurs in C_confl then
        C_reason ← reason(ℓ, ρ);
        C_reason ← reduceSat(C_reason, C_confl, ℓ, ρ);
        C_confl ← resolve(C_confl, C_reason, ℓ);
        C_confl ← saturate(C_confl);
    end
    ρ ← removeLast(ρ);
end
return C_confl;
```

Figure 7.17: Pseudo-Boolean conflict analysis analyzePBconflict(C_{confl}, ρ).

nation of two constraints C and D, then it is not hard to verify that

$$slack(c \cdot C + d \cdot D; \rho) \geq c \cdot slack(C; \rho) \; + \; d \cdot slack(D; \rho) \tag{7.46}$$

holds. By the invariant, we know for the currently derived constraint C_{confl} on the conflict side that we have $slack(C_{\text{confl}}; \rho) < 0$. It is also easy to see directly from the definition (7.38) that weakening the reason constraint C_{reason} leaves $slack(C_{\text{reason}}; \rho)$ unchanged, since we only weaken on non-falsified literals. But saturation can decrease the slack, and if we have not reached non-positive slack before, then at the very latest this will happen when all non-falsified literals except the propagating one have been weakened away — at this stage the only coefficient contributing to the slack is that of the propagating literal, and since the constraint is saturated this coefficient must be equal to the degree of falsity, so that the whole constraint has slack 0. (This is exactly what happened in our example.) Plugging this into (7.46), we see that a positive linear combination of zero and a negative number will be negative, and the invariant is maintained.

Using this reason reduction method, the whole pseudo-Boolean conflict analysis algorithm will be as in Figure 7.17. The reduction step highlighted in boldface is new compared to CDCL, but everything else is essentially the same (at least at a high level). So how does our conflict analysis compare to CDCL? Let us just point out three important aspects here, which will motivate some of the discussions later.

1. One difference is how much work needs to be performed at each step. When we resolve a new reason with the current conflict clause in CDCL, then we only have to "tag on" the reason clause to the conflict clause, but we do not have to touch the literals already in the conflict clause. Therefore, the total amount of work during CDCL conflict analysis is linear in the sum of the clause sizes. But in pseudo-Boolean analysis we might have to multiply both the reason and the conflict in order to adjust the coefficients so that the linear combination in (7.33) is cancelling, and this means that we might have to touch all literals in the constraint on the conflict side over and over again. In the worst case, this will incur an extra linear factor in the running time
2. Because of these multiplications, it can also be the case that the coefficients

in the constraints grow very large. If this happens, then the integer arithmetic can get hugely expensive. This can become a very serious problem in practice.

3. Perhaps the most serious problem, though, is that for inputs in CNF this procedure described above degenerates to resolution. All that will happen during the conflict analysis is that trivial resolution derivations will produce new clauses, and so the whole algorithm just becomes CDCL but with much more expensive data structures. Hence, if we take the pigeonhole principle formula encoded in CNF and feed it into a pseudo-Boolean solver using this conflict analysis, then although the formula is easy for cutting planes (as shown in [CCT87]) it will be exponentially hard in practice.

The third issue is perhaps the most important one in that it shows how sensitive pseudo-Boolean solvers can be to details of the encoding. We will return to the question of CNF inputs and discuss it in a bit more detail in Section 7.7.

7.6.2.3. Pseudo-Boolean Conflict Analysis Using Division

Motivated by the above discussion, it seems interesting to consider whether the division rule could be a competitive alternative to saturation. It is known that general cutting planes (consisting of the rules (7.27a)–(7.27c) and (7.34)) is *implicationally complete*, meaning that if a pseudo-Boolean formula implies a certain constraint, then there is a way to derive this constraint.[38] This is not true for cutting planes with saturation [VEG+18], i.e., when the saturation rule (7.36b) is substituted for the division rule (7.34). For instance, it can be shown that there is no way to derive the cardinality constraint

$$x + y + z \geq 2 \tag{7.47}$$

from the (translation into pseudo-Boolean constraints of the) CNF encoding

$$x + y \geq 1 \tag{7.48a}$$
$$x + z \geq 1 \tag{7.48b}$$
$$y + z \geq 1 \tag{7.48c}$$

using cutting planes with saturation (even with unrestricted linear combinations). In view of this, it is natural to ask whether the use of division could perhaps yield a stronger conflict analysis algorithm. As mentioned above, using division was proposed in the context of general integer linear programming in *CutSat* [JdM13], although it appears that this approach does not work so well in practice. What we will discuss below is a fairly recent variant of pseudo-Boolean conflict analysis that uses division instead of saturation, and that does seem to perform very well in practice.

In the division-based conflict analysis method, for each step in the conflict analysis we will iterate the following three actions

[38] This is the analogue of Theorem 7.3.1 for resolution saying that of a clause C is implied by a CNF formula F, then there is a resolution derivation of some clause $C' \subseteq C$ from F.

$c \leftarrow \mathit{coeff}(C_{\text{reason}}, \ell)$;
while $\mathit{slack}(\mathsf{resolve}(C_{\text{confl}}, \mathsf{divide}(C_{\text{reason}}, c), \ell); \rho) \geq 0$ **do**
  $\ell' \leftarrow$ literal in $C_{\text{reason}} \setminus \{\ell\}$ such that $\overline{\ell}_j \notin \rho$ and $c \nmid \mathit{coeff}(C, \ell')$;
  $C_{\text{reason}} \leftarrow \mathsf{weaken}(C_{\text{reason}}, \ell')$;
end
return $\mathsf{divide}(C_{\text{reason}}, c)$;

Figure 7.18: Reduction $\mathsf{reduceDiv}(C_{\text{confl}}, C_{\text{reason}}, \ell, \rho)$ using division.

1. Weaken the reason constraint C_{reason} on some non-falsified literal ℓ' with coefficient not divisible by the coefficient of the propagating literal ℓ to obtain $\mathsf{weaken}(C_{\text{reason}}, \ell')$.
2. Divide the weakened constraint by the coefficient c of the propagating literal, yielding the constraint $\mathsf{divide}(\mathsf{weaken}(C_{\text{reason}}, \ell'), c)$.
3. Compute the resolvent $\mathsf{resolve}(C_{\text{confl}}, \mathsf{divide}(\mathsf{weaken}(C_{\text{reason}}, \ell'), c), \ell)$ of this constraint with the conflicting constraint C_{confl} over the propagated literal ℓ.

These steps are iterated until the resolvent obtained is conflicting. (Again, the constraint C_{reason} is updated to $\mathsf{weaken}(C_{\text{reason}}, \ell')$ after each step.)

Before arguing about correctness, let us do as we did for the saturation-based method and illustrate how this approach works for the constraints C_1 and C_2 from (7.41a)–(7.41b) with the trail $\rho_3 = \big(x_1 \xleftarrow{\mathsf{DEC}} 0, x_2 \xleftarrow{C_1} 1, x_3 \xleftarrow{C_1} 1\big)$ in (7.42c), under which C_2 is conflicting. The first attempt to resolve the reason C_1 for x_3 with the conflict constraint C_2 now yields the derivation

$$\dfrac{\dfrac{\dfrac{2x_1 + 2x_2 + 2x_3 + x_4 \geq 4}{2x_1 + 2x_2 + 2x_3 \geq 3}\,\text{Weakening on } x_4}{x_1 + x_2 + x_3 \geq 2}\,\text{Division by } 2 \qquad 2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 3}{0 \geq 1}\,\text{Resolution on } x_3 \tag{7.49}$$

so for this particular example the solver immediately derives contradiction and can terminate with a report that the instance is unsatisfiable!

The pseudo-code for the reason reduction algorithm from [EN18] using division is given in Figure 7.18. Let us sketch the argument why this algorithm is guaranteed to return a reduced reason constraint that will maintain our desired invariant, namely that the resolvent of this constraint with the current constraint on the conflict-side has negative slack with respect to the current trail. Just as in the analysis of the reason reduction algorithm reduceSat using saturation in Figure 7.16, it is sufficient to prove that at some point the slack of the constraint on the reason side is guaranteed to become non-positive. This is sufficient to maintain the conflict analysis invariant of negative slack, since the constraint C_{confl} on the conflict side has negative slack by assumption and slack is subadditive (7.46), meaning that the resolvent of the reason and conflict constraints also has to have negative slack.

Following the notation in Figure 7.18, let c be the coefficient of the literal ℓ propagated by C_{reason}. By the definition of propagation in (7.40) we have

$$0 \leq \mathit{slack}(C_{\text{reason}}; \rho) < c \ , \tag{7.50}$$

```
c ← coeff(C, ℓ);
foreach literal ℓ′ in C do
    if ℓ̄′ ∉ ρ and c ∤ coeff(C, ℓ′) then
        C ← weaken(C, ℓ′);
    end
end
return divide(C, c);
```

Figure 7.19: Reason reduction roundToOne(C, ℓ, ρ) used in *RoundingSat*.

```
while C_confl contains no or multiple falsified literals on last level do
    if no current solver decisions then
        output UNSAT and terminate
    end
    ℓ ← literal assigned last on trail ρ;
    if ℓ̄ occurs in C_confl then
        C_confl ← roundToOne(C_confl, ℓ̄, ρ);
        C_reason ← roundToOne(reason(ℓ, ρ), ℓ, ρ);
        C_confl ← resolve(C_confl, C_reason, ℓ);
    end
    ρ ← removeLast(ρ);
end
ℓ ← literal in C_confl last falsified by ρ;
return roundToOne(C_confl, ℓ, ρ);
```

Figure 7.20: Pseudo-Boolean conflict analysis in *RoundingSat* using roundToOne.

and since weakening on non-falsified literals does not change the slack these inequalities hold at all times during the execution of reduceDiv. Suppose that we have reached the point in the algorithm when all coefficients of non-falsified literals not divisible by c have been weakened away. Consider what contribution the literals in $\mathsf{divide}(C_{\text{reason}}, c)$ give to the slack. Falsified literals in C_{reason} do not contribute at all, and all remaining non-falsified literals have coefficients divisible by c. Therefore, the slack of the reason constraint is divisible by c, i.e., we have

$$\mathit{slack}(\mathsf{divide}(C_{\text{reason}}, c); \rho) = \frac{\mathit{slack}(C_{\text{reason}}; \rho)}{c} \ , \tag{7.51}$$

and it follows from this and (7.50) that

$$0 \leq \mathit{slack}(\mathsf{divide}(C_{\text{reason}}, c); \rho) < 1 \ , \tag{7.52}$$

i.e., $\mathit{slack}(\mathsf{divide}(C_{\text{reason}}, c); \rho) = 0$. This proves the correctness of reduceDiv.

We remark that the reason reduction method roundToOne used in [EN18], and presented in Figure 7.19, does not weaken away literals one by one, but does the maximal amount of weakening right away. This is guaranteed to maintain the invariant by the proof just outlined above. Also, this method is used not only for reason reduction but is applied more aggressively during the conflict analysis. The pseudo-code for the conflict analysis in *RoundingSat* is presented in Figure 7.20.

It is an interesting question how saturation and division compare when used for pseudo-Boolean solving, but this is currently not very well understood. It is clear from our example (7.49) that division can sometimes be more efficient, but one can also construct crafted benchmarks where it seems that saturation should have the potential to work better [GNY19]. Nevertheless, some preliminary conclusions are that for instances where pseudo-Boolean reasoning does not help, so that a competitive approach would have been to translate to CNF and run a CDCL solver, then the conflict speed, and hence the search speed, is orders of magnitude faster in *RoundingSat* than in *Sat4j* [EN18]. For crafted benchmarks where pseudo-Boolean reasoning appears to be crucial, the number of generated conflicts per second in *RoundingSat* goes down significantly, but the performance is still much better than for pseudo-Boolean solvers using saturation [EGNV18]. One extra bonus is that the frequent use of division helps keep the coefficients small, so that one can use fixed-precision arithmetic (this of course also assumes that there is code for handling overflow, which is an issue we ignore in the pseudo-code presented here). However, the main problem we identified with saturation-based solvers still remains: for CNF inputs, the algorithm degenerates to a CDCL search for resolution proofs.

7.6.3. More Pseudo-Boolean Rules

Before wrapping up our discussion of pseudo-Boolean solving, we wish to mention some other reasoning rules that are relevant to consider in this context. A natural question to ask is whether general linear constraints are needed to harness the full power of pseudo-Boolean solvers, or whether solvers could equally well work with a more limited set of constraints. One particularly interesting class of linear inequalities are cardinality constraints (7.26). Given a general PB constraint such as, for instance,

$$3x_1 + 2x_2 + x_3 + x_4 \geq 4 \ , \tag{7.53a}$$

one can compute the least number of literals that have to be true, which results in the constraint

$$x_1 + x_2 + x_3 + x_4 \geq 2 \ . \tag{7.53b}$$

This is used in the solver *Galena* [CK05], which only learns cardinality constraints. The fact that all learned constraints will be of a particular form can also make other aspects of the algorithm easier. Formally, this *cardinality constraint reduction* rule can be written as

$$\frac{\sum_{i\in[n],\,\sigma\in\{0,1\}} a_i^\sigma x_i^\sigma \geq A}{\sum_{(i,\sigma)\,:\,a_i^\sigma>0} x_i^\sigma \geq T} \quad T = \min\Big\{|I| \,:\, I\subseteq[n],\, \textstyle\sum_{i\in I,\,\sigma} a_i^\sigma \geq A\Big\} \ . \tag{7.54}$$

Another interesting rule is *strengthening*, which we also introduce by giving an example. Consider again the constraints (7.48a)–(7.48c) and suppose that we set $x \xleftarrow{\mathsf{DEC}} 0$ and propagate. This yields $y \overset{x+y\geq 1}{\longleftarrow} 1$ and $z \overset{x+z\geq 1}{\longleftarrow} 1$, meaning that the final constraint $y + z \geq 1$ is "oversatisfied" by a margin of 1. A closer analysis of this situation leads to the conclusion that we can deduce the constraint $x + y + z \geq 2$ in (7.47), since either x is true, in which case the constraint certainly holds, or else $y + z \geq 1$ is oversatisfied as we just saw. Slightly more formally, the *strengthening*

rule, which seems to have been imported by [DG02] from operations research, can be described as follows:

- Suppose that assigning $x_j^{\sigma_j} = 0$ implies that the constraint $\sum_{i \neq j,\, \sigma} a_i x_i^\sigma \geq A$ has to be oversatisfied by an amount of K.
- Then it is sound to deduce the constraint

$$K x_j^{\sigma_j} + \sum_{i \neq j,\, \sigma} a_i x_i^\sigma \geq A + K \ . \tag{7.55}$$

In theory, using strengthening can allow the solver to recover from bad encodings such as CNF (in our example, we recovered the cardinality constraint (7.47) from the CNF encoding (7.48a)–(7.48c)). In practice, however, it seems hard to get this to work in an efficient way.

A final interesting scenario that we want to discuss is the following. If we have the PB constraints

$$2x + 3y + 2z + w \geq 3 \tag{7.56a}$$
$$2\overline{x} + 3y + 2z + w \geq 3 \tag{7.56b}$$

then by eyeballing we can conclude that

$$3y + 2z + w \geq 3 \tag{7.56c}$$

must hold, since x is either true or false and so can contribute to at most one of the constraints (7.56a) and (7.56b). But an application of the generalized resolution rule on these two constraints instead results in the constraint

$$6y + 4z + 2w \geq 4 \ , \tag{7.56d}$$

reflecting that this rule also takes the possibility $x = \frac{1}{2}$ into consideration. Applying saturation to (7.56d) yields

$$4y + 4z + 2w \geq 4 \tag{7.56e}$$

and division does not help either since it yields the equivalent constraint

$$2y + 2z + w \geq 2 \ , \tag{7.56f}$$

which is clearly weaker than (7.56c). As observed by [Goc17], it would be quite convenient to have an implementation of what we can call a *fusion resolution* rule

$$\frac{a_j x_j + \sum_{i \neq j,\, \sigma} b_i^\sigma x_i^\sigma \geq B \qquad a_j \overline{x}_j + \sum_{i \neq j,\, \sigma} b_i^\sigma x_i^\sigma \geq B'}{\sum_{i \neq j,\, \sigma} b_i^\sigma x_i^\sigma \geq \min\{B, B'\}} \ . \tag{7.57}$$

The need for such a rule shows up in some tricky benchmarks in [EGNV18], but we see no obvious way for cutting planes to perform such reasoning in an efficient manner.

7.6.4. Some Challenges for Pseudo-Boolean Solving

We conclude our review of conflict-driven pseudo-Boolean solving with a discussion of some of the challenges that lie ahead. On the theory side, one difficulty is that there seem to be many more degrees of freedom in PB solving compared to conflict-driven clause learning (CDCL). As we have seen above, there are several different ways of generalizing CDCL to a pseudo-Boolean context, and it seems far from obvious what is the best way to do so.

An example of an interesting question concerning the algorithms we have discussed so far is how much the reason should be weakened in the reduction step. Is it better to weaken iteratively, literal by literal, until the first point in time when the resolvent is conflicting? Or is it better to do as in *RoundingSat* and do all the weakening right away? Another curious difference from CDCL is that sometimes the slack on the conflict side can be so negative that it is possible to just skip a resolution step and still maintain that the conflict-side constraint when the last propagated literal is removed from the trail (so that its status changes from falsified to non-falsified and it starts contributing to the slack). In such a scenario, is it better to skip the resolution step, in order to get fewer applications of the resolution rule over all, and get a more "compact explanation" of the conflict, or is it preferable to always resolve? It seems that one can cook up crafted benchmarks supporting both approaches. Different aspects of pseudo-Boolean conflict analysis have been studied in [LMMW20, LMW20], but there is room for much more work in this area.

This leads to the more general question of whether there is a better approach for conflict analysis than generating (the analogue of) trivial resolution derivations. Note that this question also makes sense for CDCL. The main reason in favour of trivial resolution seems to be that it is simple and runs very fast. But perhaps it could sometimes pay off to be slower and do something smarter? Or, in the opposite direction, could it be that one should not try to learn general pseudo-Boolean constraints, as described above, but instead focus on a more limited form such as cardinality constraints, as done in *Galena* [CK05]?

One reason that we do not have any good answers to these questions is that we do not know too much about how the different subsystems of cutting planes described towards the end of Section 7.6.1 relate to each other in terms of theoretical strength. Some progress has been made in recent papers such as [VEG+18, GNY19], but many open problems still remain. A particularly intriguing question is whether cutting planes with division is strictly stronger than cutting planes with saturation when the linear combination rule is limited to generalized resolution. We will discuss this further in Section 7.7.4.

Among the implementation challenges, one of the most important ones is how to achieve efficient propagation for pseudo-Boolean constraints. For CDCL solvers, a simple but crucial observation is that as long as a disjunctive clause contains two non-falsified literals it cannot propagate, and the famous *two-watched-literals scheme* implementing this approach in state-of-the-art solvers is an important part of the explanation how such solvers can run so fast. This is not true in pseudo-Boolean solving — for a constraint like $\sum_{i=1}^{n} x_i \geq n-1$ one has to watch all variables in order to detect propagation, since as soon as any single variable

is set to false all others should propagate to true. The recent paper [Dev20] provides what is currently the state of the art in pseudo-Boolean propagations and discusses in more detail the challenges that remain.

A second major challenge is if and how natural and efficient methods can be designed to recover from bad encodings (such as CNF). Although it is hard to make anything like this into a formal claim, it seems that pseudo-Boolean solvers are more sensitive to the exact encoding of a problem, and also to the presence or absence of extra, potentially superfluous constraints, than CDCL solvers. A recent work reporting progress on implementing efficient pseudo-Boolean search for CNF formulas, assuming that these formulas encode cardinality constraints or that such constraints can play an important part in the proofs, is [EN20], but much work still remains.

Another interesting question is how far the solver should backjump once an asserting constraint has been learned. In contrast to CDCL, a learned constraint can be asserting at several levels, and there are even different options in how to define what an asserting constraint should be depending on whether one wants to apply syntactic or (potentially harder to check) semantic criteria. A somewhat related concern is how to assess the "quality" of different constraints, for instance, when the solver has the choice of either performing or skipping a resolution step, and one would like to know which of the two options seems to yield the better constraint.

A puzzling observation, made in, e.g., [EGNV18], is that sometimes pseudo-Boolean solvers perform extremely poorly on instances which are infeasible even viewed as linear program relaxations without any integrality constraints. Such instances can be trivial for *mixed integer linear programming (MIP)* solvers such as *Gurobi* [Gur], *CPLEX* [CPL], and *SCIP* [SCI], which will detect infeasibility after solving the LP at the root node, while the pseudo-Boolean solvers get completely lost looking for satisfying Boolean assignments in a search space where there are not even any real-valued solutions. On the other hand, it seems that sometimes MIP solvers can fail badly on instances where learning from failed partial assignments to PB constraints appears to be crucial (and where, for this reason, conflict-driven PB solvers can shine). For this reason, a quite tempting proposition would be to borrow techniques from, or merge techniques with, MIP solvers to obtain pseudo-Boolean solvers that could provide the best of both worlds. Some research in this direction has been initiated in [DGN20], but again it seems that most of the work is still ahead.

In order to further develop pseudo-Boolean solving techniques, however, a final challenge that we wish to highlight is that it would be very desirable to develop efficient and concise pseudo-Boolean proof logging techniques. Although the DRAT format employed for CDCL solvers could also be used to log pseudo-Boolean proofs in theory, in practice the required overhead makes this infeasible with current techniques. It seems that the focus on logging only disjunctive clauses mixes very poorly with what is arguably the main advantage of pseudo-Boolean solvers, namely the stronger format of the constraints they derive. There is some recent work on a proof checker *VeriPB* [GMN20b, Ver19] for pseudo-Boolean reasoning, which has been shown to admit efficient proof logging for some constraint programming techniques [EGMN20] and graph solving

algorithms [GMN20a, GMM+20], but there is currently no version that would support proof logging for the full range of techniques used by PB solvers in, e.g., the latest pseudo-Boolean solver competition [Pse16].

7.7. Cutting Planes and Proof Complexity

In the previous section we defined different flavours of the cutting planes proof system, and then showed how the conflict-driven SAT solving framework could be lifted from clauses to general 0-1 integer linear constraints. We now wish to proceed to discussing what is known about cutting planes from the point of view of proof complexity.

Before doing so, however, it will be helpful to describe a particular way of constructing crafted benchmarks that has played a crucial role in many of the recent advances on the theoretical side, namely *lifting*. The formal definition of lifted formulas seems to have appeared first in [BHP10], though our discussion below relies heavily on the later paper [HN12]. Readers more interested in seeing statements of concrete proof complexity results can safely skip ahead to Section 7.7.2.

7.7.1. Brief Detour: Lifted CNF Formulas

The idea behind lifting of functions is that we can take a base function f over some domain (which we will choose to be Boolean in this discussion) and extend it to a function over tuples from the same domain by combining it with an *indexing* or *selector* function that determines on which coordinates from the tuples f should be evaluated. More formally, given a positive integer $\ell \geq 2$ and a function $f : \{0,1\}^m \to Q$ for some range Q, the *lift of length ℓ of f* is the function $\mathit{Lift}_\ell(f) : \{0,1\}^{m\times\ell} \times [\ell]^m \to Q$ such that for any bit-vector $\{x_{i,j}\}_{i\in[m],j\in[\ell]}$ and any $y \in [\ell]^m$ the value of the lifted function is

$$\mathit{Lift}_\ell(f)(x,y) = f(x_{1,y_1}, x_{2,y_2}, \ldots, x_{m,y_m}) \ . \tag{7.58}$$

In words, the y-vector selects which coordinates of the x-vector should be fed to f, as illustrated in Figure 7.21. We refer to the coordinates of the y-vector as *selector variables* and the coordinates of the x-vector as *main variables*, and we write

$$\mathsf{select}_y(x) = (x_{1,y_1}, x_{2,y_2}, \ldots, x_{m,y_m}) \tag{7.59}$$

to denote the substring of the x-vector selected by the y-coordinates. (With this notation, we have $\mathsf{select}_y(x) = (x_{1,3}, x_{2,1}, x_{3,2}, x_{4,2})$ in Figure 7.21.)

We next extend this definition from functions to *relations*, or *search problems*. To this end, let S be any search problem defined as a subset of $\{0,1\}^m \times Q$; that is, on any input $a \in \{0,1\}^m$, the problem is to find some $q \in Q$ such that $(a,q) \in S$. Then we define the *lift of length ℓ of S* to be a new search problem $\mathit{Lift}_\ell(S) \subseteq \{0,1\}^{m\times\ell} \times [\ell]^m \times Q$ with input domain $\{0,1\}^{m\times\ell} \times [\ell]^m$ and output range Q such that for any bit-vector $\{x_{i,j}\}_{i\in[m],j\in[\ell]}$, any $y \in [\ell]^m$, and any $q \in Q$, it holds that

$$(x,y,q) \in \mathit{Lift}_\ell(S) \quad \text{if and only if} \quad \big(\mathsf{select}_y(x), q\big) \in S \ . \tag{7.60}$$

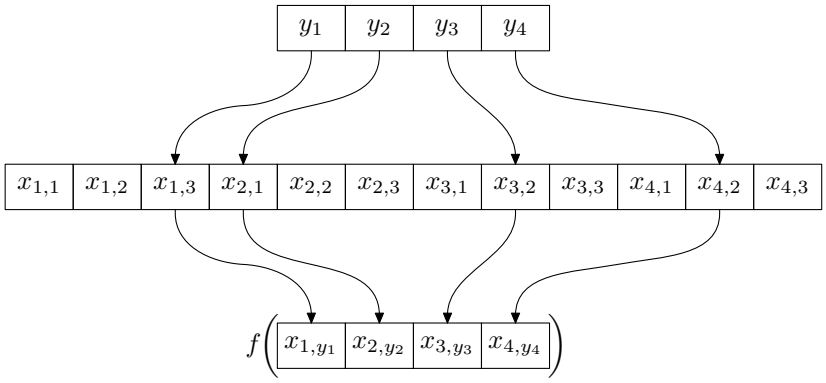


Figure 7.21: Illustration of lifted function (for $\ell = 3$, $m = 4$, and $y = (3, 1, 2, 2)$).

The general paradigm of lifting is to prove that if the function f or the search problem S is hard to compute in some weak computational model, then applying lifting produces a new function $\mathit{Lift}_\ell(f)$ or search problem $\mathit{Lift}_\ell(S)$ that is hard for a much stronger computational models (in which the original problems f or S might be trivial). This is sometimes referred to as *hardness escalation*.

The key behind several recent results establishing lower bounds for the cutting planes proof system has been to study lifted search problems defined in terms of CNF formulas using tools from the area of *communication complexity* [KN97, RY20]. Such lifted search problems are not themselves CNF formulas syntactically speaking, however, and therefore an additional step is needed where these lifted search problems are encoded back into CNF. Such *lifted CNF formulas*, as first introduced in [BHP10], can be constructed in the following way.

Definition 7.7.1 (Lift of CNF formula [BHP10])**.** Given any CNF formula F with clauses $C_1, \ldots, C_m$ over variables $u_1, \ldots, u_n$, and any positive integer $\ell \geq 2$, the *lift of length ℓ of F* is a CNF formula $\mathit{Lift}_\ell(F)$ over $2\ell n$ variables denoted by $\{x_{i,j}\}_{i\in[n],j\in[\ell]}$ (*main variables*) and $\{y_{i,j}\}_{i\in[n],j\in[\ell]}$ (*selector variables*), consisting of the following clauses:

- For every $i \in [n]$, an *auxiliary clause*

$$y_{i,1} \vee y_{i,2} \vee \cdots \vee y_{i,\ell} \ . \tag{7.61a}$$

- For every clause $C_i \in F$, where $C_i = u_{i_1} \vee \cdots \vee u_{i_s} \vee \overline{u}_{i_{s+1}} \vee \cdots \vee \overline{u}_{i_{s+t}}$ for some $i_1, \ldots, i_{s+t} \in [n]$, and for every tuple $(j_1, \ldots, j_{s+t}) \in [\ell]^{s+t}$, a *main clause*

$$\overline{y}_{i_1,j_1} \vee x_{i_1,j_1} \vee \cdots \vee \overline{y}_{i_s,j_s} \vee x_{i_s,j_s} \vee \overline{y}_{i_{s+1},j_{s+1}} \vee \overline{x}_{i_{s+1},j_{s+1}} \vee \cdots \vee \overline{y}_{i_{s+t},j_{s+t}} \vee \overline{x}_{i_{s+t},j_{s+t}} \tag{7.61b}$$

(where we will refer to C_i as the *original clause* corresponding to the *lifted clause* in (7.61b)).

Let us try to decipher what the notation in Definition 7.7.1 means. The purpose of the auxiliary clauses in (7.61a) is to make sure that in every variable subset $\{y_{i,j} \mid 1 \leq j \leq \ell\}$ at least one of the variables is true. We can think of the selector variables as encoding the vector $y \in [\ell]^m$ in the lifted search problem above. Since every pair $\overline{y}_{i,j} \vee x_{i,j}$ in a main clause (7.61b) is equivalent to an implication $y_{i,j} \to x_{i,j}$, we can rewrite (7.61b) as

$$(y_{i_1,j_1} \to x_{i_1,j_1}) \vee \cdots \vee (y_{i_{s+t},j_{s+t}} \to \overline{x}_{i_{s+t},j_{s+t}}) \ . \tag{7.62}$$

Now we can see that for every clause C_i, the auxiliary clauses encode that there is some choice of selector variables $y_{i,j}$ which are all true, and for this choice of selector variables the $x_{i,j}$-variables in the lifted clause give us back the original clause C_i. It is easily verified that F is unsatisfiable if and only if $G = \mathit{Lift}_\ell(F)$ is unsatisfiable, and that if F is a k-CNF formula with m clauses, then G is a $\max(2k, \ell)$-CNF formula with at most $m\ell^k + n$ clauses. In Figure 7.22 we show the lifted formula $\mathit{Lift}_2\big(\mathit{Peb}_{\Pi_2}\big)$ of length 2 obtained from the pebbling contradiction in Figure 7.11b.[39]

We remark that there are also other types of lifted CNF formulas in the literature, but Definition 7.7.1 (which, formally speaking, describes lifting using the *indexing gadget*), is the most commonly used version. In the most general sense, though, the pebbling formula with XOR-substitution in Figure 7.12 is also a form of lifted formula.

7.7.2. Cutting Planes Size, Length, and Space

Returning from the detour and focusing again on the cutting planes proof system, the *length* of a cutting planes refutation is the total number of lines/inequalities in it, and the *size* also sums the sizes of all coefficients (i.e., the bit size of representing them). The natural generalization of clause space in resolution is to define *(line) space* as the maximal number of linear inequalities needed in memory during a refutation, since every clause is translated into a linear inequality. There is no useful analogue of width/degree known for cutting planes.[40]

Cutting planes can simulate resolution efficiently with respect to length/size and space simultaneously by mimicking the resolution steps one by one. Hence, just as for polynomial calculus we can conclude that the worst-case upper bounds for resolution carry over to cutting planes.

The cutting planes proof system is strictly stronger than resolution with respect to length and size, since cutting planes can refute pigeonhole principle formulas (7.7a)–(7.7b) efficiently [CCT87]. The reason for this is that in contrast to resolution (and polynomial calculus), cutting planes can count. At a high level, pigeonhole principle formulas are refuted simply but summing up the number of

[39] A very closely related CNF formula, but complemented with clauses $\overline{y}_{i,j} \vee \overline{y}_{i,j'}$ for $1 \leq j < j' \leq \ell$ to enforce that assignments satisfying clauses (7.61a) set only one variable $y_{i,j}$ to true, can be generated in DIMACS format by the tool *CNFgen* [LENV17, CNF] with the command line `cnfgen peb pyramid` ⟨`h`⟩ `-T lift 2` (for pyramid graphs of height h). Whether these extra clauses are added or not typically does not affect the proof complexity bounds we can obtain for lifted formulas.

[40] That is, one can certainly define width measures, but no such measure is known to have any interesting relation to other complexity measures for cutting planes.

$$
\begin{aligned}
&(y_{u,1} \vee y_{u,2}) \\
\wedge\ &(y_{v,1} \vee y_{v,2}) \\
\wedge\ &(y_{w,1} \vee y_{w,2}) \\
\wedge\ &(y_{x,1} \vee y_{x,2}) \\
\wedge\ &(y_{y,1} \vee y_{y,2}) \\
\wedge\ &(y_{z,1} \vee y_{z,2}) \\
\wedge\ &(\overline{y}_{u,1} \vee x_{u,1}) \\
\wedge\ &(\overline{y}_{u,2} \vee x_{u,2}) \\
\wedge\ &(\overline{y}_{v,1} \vee x_{v,1}) \\
\wedge\ &(\overline{y}_{v,2} \vee x_{v,2}) \\
\wedge\ &(\overline{y}_{w,1} \vee x_{w,1}) \\
\wedge\ &(\overline{y}_{w,2} \vee x_{w,2}) \\
\wedge\ &(\overline{y}_{u,1} \vee \overline{x}_{u,1} \vee \overline{y}_{v,1} \vee \overline{x}_{v,1} \vee \overline{y}_{x,1} \vee x_{x,1}) \\
\wedge\ &(\overline{y}_{u,1} \vee \overline{x}_{u,1} \vee \overline{y}_{v,1} \vee \overline{x}_{v,1} \vee \overline{y}_{x,2} \vee x_{x,2}) \\
\wedge\ &(\overline{y}_{u,1} \vee \overline{x}_{u,1} \vee \overline{y}_{v,2} \vee \overline{x}_{v,2} \vee \overline{y}_{x,1} \vee x_{x,1}) \\
\wedge\ &(\overline{y}_{u,1} \vee \overline{x}_{u,1} \vee \overline{y}_{v,2} \vee \overline{x}_{v,2} \vee \overline{y}_{x,2} \vee x_{x,2}) \\
\wedge\ &(\overline{y}_{u,2} \vee \overline{x}_{u,2} \vee \overline{y}_{v,1} \vee \overline{x}_{v,1} \vee \overline{y}_{x,1} \vee x_{x,1}) \\
\wedge\ &(\overline{y}_{u,2} \vee \overline{x}_{u,2} \vee \overline{y}_{v,1} \vee \overline{x}_{v,1} \vee \overline{y}_{x,2} \vee x_{x,2}) \\
\wedge\ &(\overline{y}_{u,2} \vee \overline{x}_{u,2} \vee \overline{y}_{v,2} \vee \overline{x}_{v,2} \vee \overline{y}_{x,1} \vee x_{x,1}) \\
\wedge\ &(\overline{y}_{u,2} \vee \overline{x}_{u,2} \vee \overline{y}_{v,2} \vee \overline{x}_{v,2} \vee \overline{y}_{x,2} \vee x_{x,2}) \\
\wedge\ &(\overline{y}_{v,1} \vee \overline{x}_{v,1} \vee \overline{y}_{w,1} \vee \overline{x}_{w,1} \vee \overline{y}_{y,1} \vee x_{y,1}) \\
\wedge\ &(\overline{y}_{v,1} \vee \overline{x}_{v,1} \vee \overline{y}_{w,1} \vee \overline{x}_{w,1} \vee \overline{y}_{y,2} \vee x_{y,2}) \\
\wedge\ &(\overline{y}_{v,1} \vee \overline{x}_{v,1} \vee \overline{y}_{w,2} \vee \overline{x}_{w,2} \vee \overline{y}_{y,1} \vee x_{y,1}) \\
\wedge\ &(\overline{y}_{v,1} \vee \overline{x}_{v,1} \vee \overline{y}_{w,2} \vee \overline{x}_{w,2} \vee \overline{y}_{y,2} \vee x_{y,2}) \\
\wedge\ &(\overline{y}_{v,2} \vee \overline{x}_{v,2} \vee \overline{y}_{w,1} \vee \overline{x}_{w,1} \vee \overline{y}_{y,1} \vee x_{y,1}) \\
\wedge\ &(\overline{y}_{v,2} \vee \overline{x}_{v,2} \vee \overline{y}_{w,1} \vee \overline{x}_{w,1} \vee \overline{y}_{y,2} \vee x_{y,2}) \\
\wedge\ &(\overline{y}_{v,2} \vee \overline{x}_{v,2} \vee \overline{y}_{w,2} \vee \overline{x}_{w,2} \vee \overline{y}_{y,1} \vee x_{y,1}) \\
\wedge\ &(\overline{y}_{v,2} \vee \overline{x}_{v,2} \vee \overline{y}_{w,2} \vee \overline{x}_{w,2} \vee \overline{y}_{y,2} \vee x_{y,2}) \\
\wedge\ &(\overline{y}_{x,1} \vee \overline{x}_{x,1} \vee \overline{y}_{y,1} \vee \overline{x}_{y,1} \vee \overline{y}_{z,1} \vee x_{z,1}) \\
\wedge\ &(\overline{y}_{x,1} \vee \overline{x}_{x,1} \vee \overline{y}_{y,1} \vee \overline{x}_{y,1} \vee \overline{y}_{z,2} \vee x_{z,2}) \\
\wedge\ &(\overline{y}_{x,1} \vee \overline{x}_{x,1} \vee \overline{y}_{y,2} \vee \overline{x}_{y,2} \vee \overline{y}_{z,1} \vee x_{z,1}) \\
\wedge\ &(\overline{y}_{x,1} \vee \overline{x}_{x,1} \vee \overline{y}_{y,2} \vee \overline{x}_{y,2} \vee \overline{y}_{z,2} \vee x_{z,2}) \\
\wedge\ &(\overline{y}_{x,2} \vee \overline{x}_{x,2} \vee \overline{y}_{y,1} \vee \overline{x}_{y,1} \vee \overline{y}_{z,1} \vee x_{z,1}) \\
\wedge\ &(\overline{y}_{x,2} \vee \overline{x}_{x,2} \vee \overline{y}_{y,1} \vee \overline{x}_{y,1} \vee \overline{y}_{z,2} \vee x_{z,2}) \\
\wedge\ &(\overline{y}_{x,2} \vee \overline{x}_{x,2} \vee \overline{y}_{y,2} \vee \overline{x}_{y,2} \vee \overline{y}_{z,1} \vee x_{z,1}) \\
\wedge\ &(\overline{y}_{x,2} \vee \overline{x}_{x,2} \vee \overline{y}_{y,2} \vee \overline{x}_{y,2} \vee \overline{y}_{z,2} \vee x_{z,2}) \\
\wedge\ &(\overline{y}_{z,1} \vee \overline{x}_{z,1}) \\
\wedge\ &(\overline{y}_{z,2} \vee \overline{x}_{z,2})
\end{aligned}
$$

Figure 7.22: Lifting of the pebbling contradiction in Figure 7.11b.

pigeons and holes, after which the observation can immediately be made that there are too many pigeons to fit into the holes. Cutting planes and polynomial calculus are incomparable with respect to size, i.e., for both proof systems one can find hard formulas that are easy for the other system. PHP formulas are an example of formulas that are hard for polynomial calculus but easy for cutting planes. In the other direction, it was recently shown in [GKRS19] that there are formulas that are easy for polynomial calculus (and even Nullstellensatz) over any field, but are hard for cutting planes. These formulas are obtained by lifting as in Section 7.7.1, but the construction is a bit too involved to describe here.

The length measure in cutting planes does not consider the size of the coefficients. It is natural to ask if, and if so how, the power of cutting planes changes when coefficients are required to be of limited size. In [BC96] it was shown that the size of the coefficients need not be larger than of exponential magnitude if one is willing to tolerate a possible polynomial blow-up in proof length. One can define a subsystem of cutting planes where all inequalities in the proofs have to have coefficients of at most polynomial magnitude measure in the input size (i.e., coefficients should be representable with a logarithmic number of bits), and this subsystem is sometimes denoted CP* in the literature. Understanding the power of CP* remains wide open.

Open Problem 7.14. Decide whether cutting planes with coefficients of polynomial magnitude (CP*) can simulate general cutting planes with at most a polynomial blow-up in proof length, or whether there are formulas for which cutting planes with unbounded coefficients is superpolynomially stronger.

When it comes to space, cutting planes is very much stronger than both resolution and polynomial calculus — it was shown in [GPT15] that any unsatisfiable CNF formula (and, in fact, any set of inconsistent 0-1 linear inequalities) can be refuted in constant line space 5 by cutting planes![41] This proof works by starting with a linear inequality/hyperplane that cuts away the all-zero point of the Boolean hypercube $\{0,1\}^n$ from the candidate list of satisfying assignments (there has to exist a clause falsified by this assignment, from which the hyperplane can be obtained), and then uses 4 auxiliary hyperplanes to remove further points $\alpha \in \{0,1\}^n$ one by one in lexicographical order until all possible assignments have been eliminated, showing that the formulas is unsatisifiable. During the course of this refutation the size of the coefficients of the hyperplanes become exponentially large, however, which the line space measure does not charge for. A very interesting question is what happens if coefficients are limited to be of polynomial magnitude, i.e., if we consider the space complexity of CP* refutations.

Open Problem 7.15. Determine whether cutting planes with coefficients of polynomial magnitude (CP*) is as strong as general cutting planes with respect to space, or whether there exist families of formulas that require superconstant space in CP*.

This question is completely open, and it cannot currently be ruled out that line space 5 would be sufficient. All that is known is that if we restrict the cutting planes proofs to have coefficients of at most constant size, then there are formulas that require $\Omega(\log\log\log n)$ space [GPT15].

If also coefficient sizes are counted, i.e., if one measures the *total space* of cutting planes refutations, then it is not hard to show a linear lower bound (for instance by combining [BW01] and [BNT13]) and a quadratic worst-case upper bound is immediately implied by resolution. For resolution this quadratic upper bound is known to be tight by [BGT14], but to the best of our knowledge no superlinear lower bounds are known on total space in cutting planes.

Proving space lower bounds, if they exist, seems challenging. It might be worth noting in this context that already cutting planes with coefficients of absolute size 2 (which is the minimum needed to simulate resolution) is quite powerful — this is sufficient to construct space-efficient refutations of pigeonhole principle formulas [GPT15] (when space is measured as the number of inequalities in memory).

For a long time, essentially the only formulas that were known to be hard for the cutting planes proof system with respect to length/size were the ***clique-coclique formulas*** (also referred to as ***clique-colouring formulas***) claiming (the negation of) that "a graph with an m-clique cannot be $(m-1)$-colourable."

[41] Recall that this means that the formula is kept on a read-only input memory, and the working memory never contain more than 5 inequalities at any given time.

The formula consists of clauses

$$q_{k,1} \vee q_{k,2} \vee \cdots \vee q_{k,n} \quad \text{[some vertex is the } k\text{th member of the clique]} \tag{7.63a}$$

$$\overline{q}_{k,i} \vee \overline{q}_{k',i} \quad \text{[clique members are uniquely defined]} \tag{7.63b}$$

$$p_{i,j} \vee \overline{q}_{k,i} \vee \overline{q}_{k',j} \quad \text{[clique members are neighbours]} \tag{7.63c}$$

$$r_{i,1} \vee r_{i,2} \vee \cdots \vee r_{i,m-1} \quad \text{[every vertex } i \text{ has a colour]} \tag{7.63d}$$

$$\overline{p}_{i,j} \vee \overline{r}_{i,\ell} \vee \overline{r}_{j,\ell} \quad \text{[neighbours have distinct colours]} \tag{7.63e}$$

where variables $p_{i,j}$ are indicators of the edges in an n-vertex graph, variables $q_{k,i}$ identify the members of an m-clique in the graph, and variables $r_{i,\ell}$ specify a colouring of the vertices, for indices ranging over $1 \leq i < j \leq n$, $1 \leq k < k' \leq m$, and $1 \leq \ell \leq m-1$.[42]

Pudlák [Pud97] proved that these formulas are hard by using a so-called *Craig interpolation* argument, specifically tailored to work for formulas with the right structure. He showed that from any short cutting planes refutation of the formula one can extract a small monotone circuit for clique, which reduces the problem to a question about size lower bounds for monotone circuits. (For completeness, we mention that essentially the same techniques were used in [HC99] to obtain exponential lower bounds for so-called ***broken mosquito screen formulas***, but due to space constraints we will not discuss these formulas further here.)

It has been conjectured that Tseitin formulas (as in Figure 7.9) should require long cutting planes refutations, since it should be hard to count mod 2 using linear inequalities. It has seemed even more likely that random k-CNF formulas should be exponentially hard. The last few years have seen some very exciting progress on cutting planes lower bounds. In a breakthrough result, exponential length lower bounds for random CNF formulas of *logarithmic* width were obtained in [HP17, FPPR17]. Unfortunately, currently it does not seem possible to apply the techniques used in these papers to formulas of constant width.

Open Problem 7.16. Prove length lower bounds for cutting planes refutations of random k-CNF formulas for some constant $k \geq 3$.

Another intriguing result established in [GGKS18] is that if one starts with k-CNF formulas (for $k = \mathrm{O}(1)$) that require large resolution width, and then applies lifting as described in Section 7.7.1, then this yields formulas which are weakly exponentially hard for cutting planes. Very recently, the cutting planes proof system was shown to be non-automatable, i.e., not to admit efficient proof search, unless $\mathsf{P} = \mathsf{NP}$ [GKMP20]. As a complement to these results indicating weaknesses of cutting planes, another recent result revealing some of the (quite unexpected) strength of this method of reasoning is that Tseitin formulas have cutting planes refutations of quasipolynomial[43] size [DT20]. Interestingly, this

[42] A closely related version of this formula, but with extra clauses $\overline{q}_{k,i} \vee \overline{q}_{k,j}$ for $1 \leq i < j \leq n$ and $\overline{r}_{i,\ell} \vee \overline{r}_{i,\ell'}$ for $1 \leq \ell < \ell' \leq m-1$, so that the assignments satisfying the clauses (7.63a) and (7.63d) must set a single variable to true, can be generated by *CNFgen* with the command line `cnfgen cliquecoloring ⟨n⟩ ⟨m⟩ ⟨m-1⟩` (for the claim that an $(m-1)$-colourable n-vertex graph can also have an m-clique). More generally, `cnfgen cliquecoloring ⟨n⟩ ⟨k⟩ ⟨c⟩` yields a formula claiming that there exists an n-vertex graph that is both c-colourable and has a k-clique. Whether the extra binary clauses discussed above are added or not typically does not affect what proof complexity bounds can be established for these formulas.

[43] That is, refutations of size $\exp\big((\log n)^{\kappa}\big)$ for some constant κ.

result was obtained by studying the proof system *stabbing planes* [BFI+18], which very roughly can be described as an extension of pseudo-Boolean reasoning where one can branch over the truth value not only of variables but of general 0-1 linear inequalities. This proof system is powerful enough to model the kind of reasoning carried out in mixed integer linear programming (MIP) solvers, and appears to be much stronger than cutting planes. However, in some settings it is possible to translate proofs from stabbing planes to cutting planes in a somewhat efficient manner, which is how the result in [DT20] is obtained.

7.7.3. Size-Space Trade-offs for Cutting Planes

Given our very limited understanding of cutting planes, it is perhaps not so surprising that not very much has known about size-space trade-offs for this proof system until quite recently.

It was shown in [GP18b] that short cutting planes refutations of a lifted version of Tseitin formulas on expanders must have large space, but this does not provide a real trade-off since it seems likely that such short refutations do not exist at all, regardless of their space complexity. Earlier, [HN12] proved that short cutting planes refutations of a lifted version of pebbling contradictions (similar to the formula in Figure 7.22) over one particular family of DAGs require large space — a result that was strengthened and generalized by [GP18b] — and for pebbling contradictions such short refutations do exist. Interestingly, and somewhat unexpectedly, all of these results follow from reductions to *communication complexity* [KN97, RY20]. The state of knowledge regarding pebbling contradictions is much worse here than for resolution and polynomial calculus, however — for the latter two proof systems we know of general methods to translate pebbling trade-offs for (essentially) arbitrary graphs into proof complexity size-space trade-offs.

Since [GPT15] established that any unsatisfiable CNF formula has a constant-space refutation, the lower bounds for pebbling contradictions in [HN12, GP18b] yield true size-space trade-off results for cutting planes, with formulas that can be refuted in both small size and small space, but where optimizing both measures simultaneously is impossible. However, the "space-efficient" refutations have coefficients of exponential size. It would be more convincing to obtain trade-offs where the small-space refutations also have small coefficients. Such results would follow if known resolution and polynomial calculus results for pebbling contradictions or Tseitin formulas over long, narrow grids could be extended also to cutting planes, since the resolution refutations establishing the upper bounds in these trade-offs can certainly be simulated by cutting planes with constant-size coefficients. It seems plausible that such trade-off results should hold for these formulas even with respect to cutting planes, but proving this has remained out of reach.

However, using lifted pebbling formulas as in Section 7.7.1, it can be proven that there are formulas which have cutting planes proofs with coefficients of constant size in either small length or small space, but for which any small-space refutation must require superpolynomial length even if the coefficients in the proof are of unbounded size. An example of such a result from [dRNV16] is that there exist CNF formulas $\{F_N\}_{N=1}^{\infty}$ of size $\Theta(N)$ such that:

- F_N can be refuted by cutting planes with constant-size coefficients in size $\mathrm{O}(N)$ and (total) space $\mathrm{O}\big(N^{2/5}\big)$.
- F_N can be refuted by cutting planes with constant-size coefficients in (total) space $\mathrm{O}\big(N^{1/40}\big)$ and size $2^{\mathrm{O}(N^{1/40})}$.
- Any cutting planes refutation of F_N, even with coefficients of unbounded size, in line space less than $N^{1/20-\epsilon}$ (for some small enough constant $\epsilon > 0$) requires length greater than $2^{\Omega(N^{1/40})}$.

In fact, this trade-off result holds simultaneously for resolution, polynomial calculus, and cutting planes, although for resolution and polynomial calculus the parameters are much worse than in the results surveyed in Sections 7.4.4 and 7.5.6.

As discussed above, determining the precise relation between general cutting planes and the subsystem CP* with coefficients of at most polynomial magnitude remains open, but what is known regarding Open problems 7.14 and 7.15 can also be stated in the form of a trade-off result. Namely, it was recently shown in [dRMN+20] that there are formulas which exhibit strong length-space trade-offs for CP* but not for general cutting planes. In more detail, it is possible to construct a family of CNF formulas $\{F_n\}_{n=1}^{\infty}$ such that:

- CP*, and even resolution, can refute F_n in length $\widetilde{O}(n)$ (this "big-O-tilde" notation hides polylogarithmic factors in n).
- Cutting planes can refute F_n in length $\widetilde{O}(n^2)$ and line space $\mathrm{O}(1)$,
- For any CP* refutation in length L and line space s it must hold that

$$s \cdot \log L = \Omega(n/\log^2 n) \ . \tag{7.64}$$

In order to make this into a "true trade-off" in the sense of [dRNV16], we would also like to show that CP* can refute these formulas in small space. This is not known, however, and in fact it is conceivable, or even seems likely, that any CP* refutation, regardless of the length, would require line space $\Omega(n/\log^2 n)$ for these formulas. Proving such a space lower bound would seem to require new techniques, however.

7.7.4. Subsystems of Cutting Planes

Recall that towards the end of Section 7.6.1 we introduced the following variants of the cutting planes proof system (following [VEG+18]), loosely corresponding to different approaches to pseudo-Boolean solving:

General cutting planes: Rules (7.27a)–(7.27c) and (7.34).

Cutting planes with resolution: As general cutting planes above, except that all applications of (7.27b) and (7.27c) have to be combined into applications of the generalized resolution rule (7.33).

Cutting planes with saturation: Rules (7.27a)–(7.27c) and (7.36b) (with no restrictions on the linear combinations).

Cutting planes with saturation and resolution: As the version of cutting planes with saturation above, except that all applications of the rules (7.27b) and (7.27c) have to be combined instances of generalized resolution (7.33).

As mentioned previously, the general cutting planes proof system is implicationally complete, meaning that if a set of linear inequalities imply some other linear inequality, then the latter inequality can also be derived syntactically. This is *not* true for the other variants of cutting planes listed above. Probably already since [Hoo88, Hoo92] it has been known that cutting planes with division and resolution collapses to the resolution proof system when the input is presented in CNF. It is not hard to show that this holds also if division is replaced by saturation (while restricting linear combinations to general resolution). Thus, subsystems of cutting planes with the generalized resolution rule rather than unrestricted linear combinations can be exponentially weaker than general cutting planes.

However, for cutting planes with saturation it turns out that the generalized resolution rule is not the problem. It was shown in [GNY19], building on [VEG+18], that if the saturation rule is used, then any refutation using unrestricted linear combinations can be transformed into a refutation using only the generalized resolution rule with at most a small polynomial blow-up in the proof length.[44] Phrased differently, for cutting planes with saturation it is not really a restriction to limit linear combinations to general resolution.

In a pseudo-Boolean solving context it is natural that the generalized resolution rule is used when taking linear combinations of constraints. Such derivation steps arise naturally during conflict analysis, as has been discussed above, and it is hard to see how to devise good heuristics for non-cancelling linear combinations (although it would be very nice if this could be done). A relevant question, therefore, is how the division and saturation rules compare to each other if cutting planes is restricted to generalized resolution. Two results in [GNY19], which we will discuss next, indicate that using division or saturation might lead to incomparable proof systems.

In one direction, there are pseudo-Boolean formulas which have linear-length refutations in cutting planes with division and resolution but require exponential-length refutations in cutting planes with saturation even if there are no restrictions on the linear combinations.

In the other direction, simulating one generalized resolution step followed by a saturation step can take an unbounded number of steps in general cutting planes with division and unrestricted linear combinations. Since the example is simple, we present it here. Let R be a positive integer and consider the two constraints

$$Rx + Ry + \sum_{i=1}^{R} z_i \geq R \tag{7.65a}$$

and

$$Rx + R\overline{y} + \sum_{i=R+1}^{2R} z_i \geq R \ . \tag{7.65b}$$

Resolving these two constraints yields

$$2Rx + \sum_{i=1}^{2R} z_i \geq R \ , \tag{7.66}$$

[44] To be technically precise, note that this is a claim about proof *length* and not about *size*. There is no guarantee in [GNY19] as to what happens to the size of the coefficients.

which after saturation becomes

$$Rx + \sum_{i=1}^{2R} z_i \geq R \ . \tag{7.67}$$

However, deriving (7.67) from (7.65a) and (7.65b) in general cutting planes (with division and arbitrary linear combinations) can be shown to require $\Omega(\sqrt{R})$ applications of the division rule. Note that this is exponential in the number of bits of the coefficients. It is important to understand that this does *not* show that cutting planes with saturation can be superpolynomially stronger than cutting planes with division — to obtain such a separation, we would need to exhibit a family of pseudo-Boolean formulas for which refutations using division have to be superpolynomially longer than refutations using saturation. What the above example does show, however, is that if CP with division and resolution polynomially simulates CP with saturation and resolution, then such a simulation is unlikely to proceed line by line and instead has to work by some kind of global argument.

Open Problem 7.17. Are there unsatisfiable pseudo-Boolean formulas for which cutting planes with saturation can produce shorter refutations than cutting planes with division?[45]

We can use the above results concerning different versions of cutting planes to understand better the potential and limitations of different approaches to pseudo-Boolean solving.

As we have already discussed, one problem with current pseudo-Boolean solvers is that they perform poorly on CNF inputs. The explanation for this is that if the input is in CNF, then the solvers cannot possibly go beyond resolution, regardless of what heuristics they use. Thus, while solvers that implement native pseudo-Boolean reasoning, such as *Sat4j* [LP10] and *RoundingSat* [EN18], can solve pigeonhole principle formulas very efficiently, they crucially depend on the input being given in pseudo-Boolean form as linear inequalities:

$$p_{i,1} + p_{i,2} + \cdots + p_{i,n} \geq 1 \qquad \text{[every pigeon } i \text{ gets a hole]} \tag{7.68a}$$

$$\overline{p}_{1,j} + \overline{p}_{2,j} + \cdots + \overline{p}_{n+1,j} \geq n \qquad \text{[no hole } j \text{ gets two pigeons]} \tag{7.68b}$$

If the input is instead presented in CNF, with the cardinality constraints in Equation (7.68b) encoded as the clauses in Equation (7.7b), then the solvers will run in exponential time. The same holds for subset cardinality formulas — if a pseudo-Boolean solver is fed the formula encoded as cardinality constraints, then it runs fast, but on the CNF version in Figure 7.10b it cannot possibly do better than the exponential lower bound on resolution length in [MN14].

An obvious algorithmic challenge is to make pseudo-Boolean solvers reason more efficiently with CNF inputs, so that they could, e.g., detect and use the cardinality constraints hidden in (7.7a)–(7.7b) to get performance comparable to

[45] Here we want the constraints in the pseudo-Boolean formula to be in saturated form, so that the separation would not be a consequence of obfuscated input, but would provide an example where reasoning with saturation during the proof search can be more efficient than reasoning with division, as in the derivation of (7.67) from (7.65a) and (7.65b).

when the input is given as (7.68a)–(7.68b). It is possible to do a preprocessing step to recover cardinality constraints encoded in CNF, and for pigeonhole principle formulas and subset cardinality formulas this works well [BLLM14], but full preprocessing of the input to try to detect cardinality constraints does not seem to be an efficient approach in general. A different idea is to try to perform cardinality constraint detection "on the fly" during the proof search, as proposed in [EN20], but although this seems like a more promising approach it, too, is not yet efficient enough to be used in a general pseudo-Boolean solver. Yet another possibility would be to develop better heuristics for solvers using division and unrestricted linear combinations, since we know that with such rules it is always possible, at least in theory, to rewrite CNF constraints in some other pseudo-Boolean form when this is desirable.

The sensitivity to the input format is not the only challenge in the context of pseudo-Boolean reasoning, however. Another challenging benchmark family are the even colouring (EC) formulas discussed in Section 7.5.4. The pseudo-Boolean encoding of these formulas is obtained by converting every equality constraint to a pair of inequalities, so that, e.g., the first constraint $u + w = 1$ in Figure 7.14b is translated to the inequalities $u + w \geq 1$ and $\overline{u} + \overline{w} \geq 1$, and the second equality $u + z = 1$ is translated to $u + z \geq 1$ and $\overline{u} + \overline{z} \geq 1$, et cetera.

If the number of edges is odd, so that the formula is unsatisfiable, then cutting planes can sum the PB constraints with positive literals over all vertices to derive $2 \cdot \sum_{e \in E(G)} e \geq |E(G)|$ and then divide by 2 and round up to obtain $\sum_{e \in E(G)} e \geq (|E(G)|+1)/2$. By instead summing up all constraints with negated literals and dividing by 2 one obtains $\sum_{e \in E(G)} \overline{e} \geq (|E(G)| + 1)/2$, and adding these two inequalities cancels all variables and leaves $0 \geq 1$.

One interesting aspect to observe here is that in contrast to pigeonhole principle and subset cardinality formulas, it is essential in the above argument that variables are integer-valued. To see the difference, suppose that we are given a PHP or subset cardinality formula encoded as linear constraints. Then for cutting planes it is sufficient to simply add up the inequalities to derive a contradiction. No integer-based reasoning is needed. Even if we allow putting fractional pigeons into fractional holes, there is no way one can make a pigeon mass of $n + 1$ fit into holes of total capacity n. This set of linear inequalities is unsatisfiable even over the rationals, i.e., the polytope defined by the constraints is empty, and so it is sufficient to just solve the linear programming relaxation. Similarly, for subset cardinality formulas there is no way $4n + 1$ variables could have a total "true mass" of at least $2n + 1$ and a total "false mass" of $2n + 1$ simultaneously. But for collections of linear constraints as in Figure 7.14b, assigning all edges value $\frac{1}{2}$ yields a satisfying fractional solution. The polytope defined by the linear inequalities is *not* empty, but it does not contain any integer points. Hence, refuting EC formulas in cutting planes crucially requires that the solvers use saturation or division (or some other form of integer-based reasoning).

Experiments in [EGNV18] show that even colouring formulas and some other crafted formulas are much harder for pseudo-Boolean solvers than the cutting planes upper bound would suggest, which seem to indicate that the solvers are still quite far from using the full power of cutting planes reasoning.[46] It also seems

[46] To balance this picture somewhat — since this survey chapter might be perceived as taking

to be the case that the division-based solver *RoundingSat* performs clearly better than *Sat4j*, and indeed it is not easy to see what a short argument using saturation for the unsatisfiability of even colouring formulas would be. It is tempting to conjecture that EC formulas generated from the right kind of graphs should be exponentially hard for cutting planes with saturation, but proving such a lower bound appears to be beyond current techniques.

Open Problem 7.18. Is it true that even colouring formulas over random 6-regular graphs with an odd number of vertices (so that the formulas are unsatisfiable) are exponentially hard for cutting planes with saturation asymptotically almost surely?

7.7.5. Further Algebraic and Semialgebraic Proof Systems

Nullstellensatz, polynomial calculus, and cutting planes have been generalized to other algebraic and semialgebraic proof systems that are more powerful in several ways. These include the *Lovász-Schrijver* proof system [LS91], the *Sherali-Adams* proof system [SA90], the *Positivstellensatz* proof system [GV01], a system combining Lovász-Schrijver and cutting planes [Pud99], and the *Lasserre* proof system [Las01a]. All of these are methods for reasoning about solutions to polynomial equations. By adapting them to prove the nonexistence of 0-1 solutions for a system of polynomials, they can be used as proof systems for refuting Boolean formulas. There has been extensive work on these systems. For an early expository account of the Positivstellensatz and the Lovász-Schrijver systems, see [GHP02a]. The survey papers [Lau01, CT12] discuss the Lovász-Schrijver, Sherali-Adams, and Lasserre systems.

The *Sum-of-Squares* proof system describes methods for establishing the (non-)existence of general solutions (not just 0-1 solutions) to systems of polynomial equalities over the real numbers. They are similar to the Positivstellensatz system. Sum-of-Squares systems were introduced by [Sho87, Nes00, Par00, Las01b]; see also the survey [BS14]. Other, very strong, algebraic proof systems are the *ideal proof system (IPS)* introduced by Grochow and Pitassi [GP18a] and the noncommutative ideal proof system of [LTW18]. An exposition of these systems is in [PT16].

A recent survey covering proof complexity and Sherali-Adams and Sum-of-Squares is given by [FKP19]. For another survey including the above topics, see [Raz16b]. Not all of the above-discussed systems are propositional proof systems in the sense of Cook and Reckhow; in particular, the validity of IPS proofs can be checked by randomized polynomial time algorithms, but no deterministic polynomial algorithm is known.

An orthogonal family of proof systems — which we mention here anyway because of the connection to cutting planes — are based on *ordered binary decision diagrams (OBDDs)*, which can provide efficient and flexible representations of Boolean functions serving as an alternative to the CNF representation

a rather dim view of the state of the art in pseudo-Boolean solving — we want to stress that there are also applications [LR18, LBD$^+$20, SDNS20] where pseudo-Boolean solvers are doing very well, and for some of these applications it seems that the power of the cutting planes proof system is crucial to get better performance than what is offered by CDCL SAT solving, algebraic approaches, constraint programming, and/or mixed integer linear programming.

(see [Bry92]). The paper [AKV04] proposed the use of OBDDs in proof systems, showing they can polynomially simulate resolution and cutting planes with small coefficients. Lower bounds on OBDD-based proof systems were shown in [AKV04], and then by [Kra08, Seg08]. Further lower bounds and simulation results have been obtained in [GZ03, TSZ10, Jär11, IKRS17, BIKS18].

7.8. Extended Resolution and DRAT Proof Systems

We now switch our focus to proof systems that are substantially stronger than the earlier considered systems of resolution, Nullstellensatz, polynomial calculus, and cutting planes. Just to give an overview of where we are going, the present section discusses DRAT-style proofs and extended resolution. These systems are directly based on resolution and CDCL solvers, and are intended for use in automated proof systems. Section 7.9 then covers Frege proof systems, which is a generalization of resolution that sits between resolution and extended resolution in proof strength. Extended resolution and Frege proof systems are both stronger than the other proof systems that we have surveyed earlier — in fact, they can give polynomial-size proofs of all the combinatorial principles and counting principles discussed earlier. The only family of formulas discussed so far for which it is not known whether they possess polynomial-size Frege and/or extended resolution proofs are random k-CNF formulas; the usual conjecture is that they do not. Section 7.10 discusses bounded-depth Frege systems, which is a restricted form of Frege systems. Bounded-depth Frege systems are weaker than full Frege systems but stronger than resolution, and are not comparable to cutting planes, in that bounded-depth Frege is more effective on some propositional formulas, but less effective on others.

Our treatment of these topics is theoretical in orientation and relatively brief, partly due to space constraints and partly since these proof systems have yet to be widely used in practical SAT solvers. The exception is that DRAT and similar systems are becoming widely used for proof logging and verification.

Extended resolution (ER) was originally introduced by Tseitin [Tse68] to allow resolution to work with more general formulas than CNFs. The intuition is that an extended resolution refutation should be allowed to introduce proof lines of the form

$$x \leftrightarrow \varphi \ , \tag{7.69}$$

where x is a *new variable* and φ is an arbitrary propositional formula. The proviso that x is new means that x does not appear in any axiom, in φ, or earlier in the derivation. The extension rule can be used in derivations as well, but then x also must not appear in the formula being derived.

The extension rule (7.69) is stated in the form that will be employed in extended Frege systems, which are defined in Section 7.9. Since resolution systems are constrained to work with clauses, *extended resolution* uses instead a restricted, clausal form of the extension rule. If a and b are literals, and x is a new variable, then the extension rule allows inferring the three clauses

$$\overline{x} \vee a \qquad \overline{x} \vee b \qquad \overline{a} \vee \overline{b} \vee x \tag{7.70}$$

which express that $x \leftrightarrow (a \wedge b)$. An *extended resolution derivation* is a derivation in which both the resolution rule (7.2) and the extension rule (7.70) are allowed. Using (7.70) multiple times, with multiple new variables, makes it possible to simulate efficiently the action of the full extension inference of (7.69). Consequently, extended resolution simulates (and is simulated by) the extended Frege proof system which uses (7.69).

It is almost immediate that adding the extension rule preserves soundness, since if there is a satisfying assignment to a set of clauses, the satisfying assignment can be extended to give the new variable x the (unique) truth value which satisfies (7.69) or (7.70). Therefore, extended resolution is sound and complete. However, resolution does not polynomially simulate extended resolution. Instead, the addition of extension variables makes the proof system exponentially more powerful. The classic example of this are the pigeonhole principle formulas, which was shown to have polynomial-size extended resolution proofs in [CR79], but to require exponential-size resolution proofs in [Hak85].

The fact that the extension rule is so powerful raises the question of whether practical, CDCL-based, SAT solvers can incorporate the extension rule. If this could be done well, the gains would be enormous, as this would in principle give CDCL the possibility to refute CNFs using the full power of extended resolution. There have been a number of attempts to do so, but the published literature on this is fairly sparse, principally [SB06, AKS10, Hua10, MHB13]. So far, the extension rule has been shown to be useful in limited situations; however, it has not been successful enough to be generally included in SAT solvers. The main bottleneck appears to be that we have no good heuristics for how to choose extension formulas φ for use in the extension rule. A second bottleneck is that even if good choices are made for extension formulas, it appears to be difficult for the CDCL solver to use the new extension variables advantageously for decision literals, unit propagation and clause learning.

In recent years, a new application for the extension rule has appeared, driven by the desire to have SAT solvers output verifiable proofs of correctness of the computations. Section 7.3.1 discussed reverse unit propagation (RUP) proofs as proof traces. RUP proofs, however, are not powerful enough to handle all the preprocessing and inprocessing techniques used by modern SAT solvers. A more powerful *resolution asymmetric tautology (RAT)* inference rule has been developed to supplant RUP, and has later been complemented with a deletion rule in the proof traces to yield the *DRAT* proof system.

The RAT rule differs from RUP rules or resolution in that it does not respect implication. That is to say, it is possible to use a RAT rule to derive a clause C from a set of clauses F even if F does not imply C. Instead, RAT inferences only preserve consistency, in that we are assured $F \cup C$ is consistent if and only if F is. This property is called *equisatisfiability*. As we shall see, examples of this include the pure literal rule and extension axioms. The pure literal rule is used to set a literal that occurs only positively to be set true; the extension axioms are allowed even though they are not true under all truth assignments. Equisatisfiability means that these steps cannot introduce an inconsistency.

Following [HHW13b, HHW13a], a *RAT inference* is formally defined as follows. Let F be the current set of clauses in the clause database. (Recall that

typically the clause database F is repeatedly updated by clause learning and clause deletion.) Let C be a clause $a_1 \vee \cdots \vee a_k \vee b$: here it is permitted, but not required, that the literal b is a new literal not appearing yet in F. Then we may infer C with a *RAT inference* with respect to b, provided that for every clause $D \vee \overline{b}$ in F it holds that the clause

$$a_1 \vee \cdots \vee a_k \vee D \vee b \tag{7.71}$$

either is tautological by virtue of containing clashing literals or is a RUP clause with respect to F. (Recall that a RUP clause is a clause C' which can be inferred by trivial resolution from F, and that this can be checked quickly by adding the negations of the literals in C' to F and using unit propagation to generate a contradiction — see Section 7.3.1 for more details).

The intuition behind a RAT inference is that a clause C of the form $a_1 \vee \cdots \vee a_k \vee b$ can be added to the clause database F without introducing a new inconsistency. One way to justify this is to prove that if the clause C is resolved in all possible ways with clauses of F using the distinguished literal b, then only clauses which are already derivable from F are obtained.

For an example of a RAT inference, let F be the set of clauses

$$\{\overline{b} \vee \overline{a}_1 \vee a_2 \vee a_3,\ \overline{b} \vee a_4,\ a_4 \vee \overline{a}_2,\ a_4 \vee \overline{a}_3,\ a_2 \vee a_3\}\ . \tag{7.72}$$

Then the clause $C = a_1 \vee b$ can be inferred by a RAT inference from F with respect to the special variable b. To verify this, note there are two clauses to consider for D by virtue of F containing $D \vee \overline{b}$. The first choice for D is $\overline{a}_1 \vee a_2 \vee a_3$; the resolvent of $D \vee \overline{b}$ with C contains both a_1 and $\overline{a}_1$ yielding a clause of the form (7.71) that is tautological. The second choice for D is a_4; now the resolvent with C is $a_1 \vee a_4$, so the clause (7.71) now becomes $a_1 \vee a_4 \vee b$. This clause is RUP with respect to F since after setting a_1, a_4 and b false in F, we can obtain the empty clause by unit propagation.

Note that the clause C of this example is *not* a consequence of F, since F is satisfied by setting b, a_1, a_4 false and a_2, a_3 true. In particular, RAT can infer C even though $F \not\models C$. Instead, the critical property is that if C can be inferred from F with a RAT inference, then F is satisfiable if and only if $F \cup \{C\}$ is satisfiable. This means that the RAT inference is sound for refutations, but not for derivations. Indeed, the resolution rule is subsumed by the RAT rule, so the RAT rule by itself yields a sound and complete refutation system.

As another example, the RAT inference subsumes the *pure literal rule*. A literal is *pure* if it occurs only positively in the clause database F; the pure literal rules allows introducing a unit clause asserting the literal is true. Specifically, for the example F of (7.72) the unit clause $\overline{a}_1$ is trivially RAT with respect to F since there is no clause containing the literal a_1. Subsequently, the unit clause $\overline{b}$ is RAT as well, for the same reason. In other words, RAT inferences can be used to infer both of the unit clauses $\overline{a}_1$ and $\overline{b}$. This is in no way contradicts the previous example, since it is not possible to introduce the clause $b \vee a_1$ at the same time, as the latter clause can no longer be inferred by a RAT inference once $\overline{a}_1$ and $\overline{b}$ have been introduced. Thus, adding clauses to F may lead to that clauses that were previously RAT clauses are no longer inferrable by a RAT inference. Conversely,

deleting clauses from F may cause clauses to be become derivable by using the RAT rule.

It is not hard to see that RAT inferences can simulate the extension rule; indeed, the three clauses $\overline{x} \vee a$, $\overline{x} \vee b$, and $\overline{a} \vee \overline{b} \vee x$ of the extension rule (7.70) can be added one at a time by RAT inferences provided F does not contain any clauses containing x or $\overline{x}$. The first two clauses can be added since F does not contain any clauses involving x; the third one, because its resolvents with the first two clauses are tautological. Conversely, it is known that extended resolution can simulate the RAT inference [KRH18]. Therefore, the DRAT proof system and extended resolution polynomially simulate each other.

These simulations between DRAT and extended resolution work because the systems allow introducing arbitrary new variables. Thus, using DRAT for proof *search* can suffer from the same difficulties as extended resolution; namely, we lack effective methods of deciding what properties the new variables should represent. A different, and very interesting, approach proposed by [HKB17, HKB20] is to consider DRAT without allowing new variables to be introduced. In recent works [HB18, KRH18, BT19], it has been shown that many of the variants of DRAT are equivalent even when new variables are not allowed to be introduced.

DRAT under the restriction of not introducing new variables is still an area of active research. It is too new to be properly surveyed here, but recent developments include [HHW15, HB18, HKSB17, BT19] and works cited in these papers. An intriguing aspect of DRAT (and related systems) is that, even when no new variables are permitted, it is still unexpectedly strong (see [HKB20, BT19]). It was shown in [BT19, HKB17, HKB20] that even without allowing new variables or clause deletion, a modest extension of the RAT rule called *subset propagation redundancy* (or *SPR* for short), first introduced in [HKB20] is powerful enough to give polynomial-size proofs of many hard examples for bounded-depth Frege proof systems, including pigeonhole principle formulas, clique-colouring formulas, and Tseitin formulas.

The paper [BT19] further showed that when new variables are not allowed, then there is an exponential separation in refutation size between the proof system RAT (the RAT rule without allowing clause deletion) and the proof system DRAT (the RAT rule combined with clause deletion). Clause deletion can give this extra power, since removing clauses permits additional RAT inferences.

An approach to automating the search for DRAT proofs was investigated in the work [HKSB17, HKB19], which used *satisfaction-driven clause learning (SDCL)* to search for short DRAT refutations without new variables. With SDSL, they were able to generate refutations of pigeonhole principle formulas and Tseitin formulas much more quickly than what CDCL can typically accomplish, finding refutations for pigeonhole principle formulas that are close to the optimal polynomial size, although still taking exponential time (at least for PHP formulas). This was accomplished in a very general way without explicitly checking for cardinality constraints. However, so far SDCL has not proved helpful in the general setting of SAT competition problems. As mentioned, there are theoretical results that deletion can exponentially increase the power of (D)RAT; however, there are at present no practical techniques for incorporating this extra power of deletion. In addition, if the deletion rule is not constrained to preserve equisatisfiability,

then deletion may change an unsatisfiable formula into a satisfiable formula. This would be incompatible with present-day CDCL solvers, which search simultaneously for satisfying assignments and proofs of unsatisfiability. For more on DRAT and related systems, see Chapter 7.11.

7.9. Frege and Extended Frege Proof Systems

Frege proofs are the standard "textbook" propositional proof systems, typically with modus ponens as the sole inference rule, and frequently formulated with connectives $\neg$, $\wedge$, $\vee$, and $\rightarrow$. In contrast to previous sections, formulas are no longer required to be in CNF, and instead can be formed using arbitrary combinations of Boolean connectives.

Frege systems were first defined in the setting of proof complexity by [CR79, Rec75]). They are axiomatized with a finite set of axiom schemes (for example $F \wedge G \rightarrow F$ is a possible axiom) and a finite set of inference rules (for instance, modus ponens). This allows for many possible Frege systems. Indeed, different sets of connectives, and different axioms and rules of inference can be used. It is required that the connectives of a Frege system can form formulas expressing any Boolean functions (that is that the connectives are "complete"); it is also required that a Frege proof system be implicationally sound and complete. To explain this last bit, let F be a formula and Γ be a set of formulas; recall that we write $\Gamma \vDash F$ to indicate that Γ logically implies F. For a Frege system $\mathcal{F}$ to be implicationally sound and complete, we must have $\Gamma \vDash F$ if and only if there is an $\mathcal{F}$-derivation of F from the formulas in Γ. As shown in [CR79, Rec75]), any two Frege systems polynomially simulate each other, so the exact choice of axioms and rules of inference is not particularly important.

We present here an alternative definition of Frege proof systems based on the sequent calculus. This is an elegant and flexible framework for formulating many different proof systems; we describe one particular version, called here LK, suitable for propositional logic.[47] *Propositional formulas* are formed using the propositional connectives $\wedge$ and $\vee$ and variables x and negated variables $\overline{x}$. Propositional formulas are defined formally by induction: firstly, for x any variable, x and $\overline{x}$ are both propositional formulas and secondly, if φ and ψ are propositional formulas, then their conjunction $(\varphi \wedge \psi)$ and their disjunction $(\varphi \vee \psi)$ are propositional formulas. Note that propositional formulas allow arbitrary use of conjunctions and disjunctions and consequently are much more general than the CNF formulas considered earlier in this survey.

The lines in an LK proof are sequents. A *sequent* is an expression of the form

$$\varphi_1, \varphi_2, \ldots, \varphi_k \Rightarrow \psi_1, \psi_2, \ldots, \psi_\ell, \tag{7.73}$$

where the φ_i's and ψ_i's are propositional formulas. The intended meaning of this sequent is that the conjunction of the formulas φ_i on the left implies the disjunction of the formulas ψ_j on the right. That is, commas on the left-hand

[47] The name LK is often used instead for a first-order logic, as originally introduced by Gentzen [Gen35]; our propositional system is sometimes called PK in the literature.

side of (7.73) should be understood as if they were $\wedge$'s, and commas on the right-hand side should be understood as if they were $\vee$'s. Equivalently, the sequent expresses that the disjunction of the ψ_j's and the $\neg\varphi_i$'s is true.

It is convenient to use the convention that in the sequent (7.73), the right- and left-hand sides are *multisets*. That is, the order of the formulas $\varphi_1, \ldots, \varphi_k$ is unimportant, but the multiplicities of formulas are tracked. We use $\Gamma, \Delta, \ldots$ to denote multisets of formulas, and often use commas to denote the union of multisets. The rules of inference for LK are:

Initial sequents: For any variable x, there are three initial sequents (*logical axioms*):

$$x \Rightarrow x \qquad \text{and} \qquad x, \overline{x} \Rightarrow \qquad \text{and} \qquad \Rightarrow x, \overline{x} \tag{7.74a}$$

Structural rules:

$$\text{Weak-left: } \frac{\Gamma \Rightarrow \Delta}{\varphi, \Gamma \Rightarrow \Delta} \qquad \text{Weak-right: } \frac{\Gamma \Rightarrow \Delta}{\Gamma \Rightarrow \Delta, \varphi} \tag{7.74b}$$

$$\text{Contract-left: } \frac{\varphi, \varphi, \Gamma \Rightarrow \Delta}{\varphi, \Gamma \Rightarrow \Delta} \qquad \text{Contract-right: } \frac{\Gamma \Rightarrow \Delta, \varphi, \varphi}{\Gamma \Rightarrow \Delta, \varphi} \tag{7.74c}$$

Cut rule:

$$\text{Cut: } \frac{\Gamma \Rightarrow \Delta, \varphi \qquad \varphi, \Gamma \Rightarrow \Delta}{\Gamma \Rightarrow \Delta} \tag{7.74d}$$

Logical rules:

$$\wedge\text{-left: } \frac{\varphi, \psi, \Gamma \Rightarrow \Delta}{\varphi \wedge \psi, \Gamma \Rightarrow \Delta} \qquad \wedge\text{-right: } \frac{\Gamma \Rightarrow \Delta, \varphi \qquad \Gamma \Rightarrow \Delta, \psi}{\Gamma \Rightarrow \Delta, \varphi \wedge \psi} \tag{7.74e}$$

$$\vee\text{-left: } \frac{\varphi, \Gamma \Rightarrow \Delta \qquad \psi, \Gamma \Rightarrow \Delta}{\varphi \vee \psi, \Gamma \Rightarrow \Delta} \qquad \vee\text{-right: } \frac{\Gamma \Rightarrow \Delta, \varphi, \psi}{\Gamma \Rightarrow \Delta, \varphi \vee \psi} \tag{7.74f}$$

For extended Frege proofs (but not for Frege or LK), *extension axioms* are also allowed as initial sequents. Extension axioms are specified by a sequence of pairs (x_i, ψ_i) for $i = 1, \ldots, k$ so that each ψ_i is a formula and each *extension variable* x_i is a "new" variable that does not appear in the conclusion of the proof and does not appear in any ψ_j for $j \leq i$. Then an extended Frege proof may use the following extension axioms as initial sequents:

Extension axioms:

$$x_i \Rightarrow \psi_i \qquad \text{and} \qquad \psi_i \Rightarrow x_i \tag{7.75}$$

An LK proof is a sequence of sequents, where each sequent either is a logical axiom or is inferred from earlier sequents by a valid LK rule of inference.[48] A *Frege proof* of a formula φ is defined to be an LK proof of the sequent $\Rightarrow \varphi$. An *extended Frege*

[48] It is typical in the literature to require LK proofs to be tree-like; we instead allow DAG-like proofs, so as to be consistent with the other systems discussed in this survey. Perhaps confusingly however, Frege proofs based on modus ponens are typically defined to be DAG-like in the literature. This makes no difference for us, as tree-like LK proofs can polynomially simulate (DAG-like) LK proofs [Kra94], so the distinction between tree-like and DAG-like proofs is not important for Frege systems.

proof of φ is defined similarly, but allowing the use of extension axioms (7.75) as initial sequents. Extended Frege is equivalent to extended resolution, in that the two systems polynomially simulate each other.

We can also use LK as a refutation system. For this, let $\mathcal{S}$ be a set of sequents. An LK refutation may use both the logical axioms of (7.74a) and sequents from $\mathcal{S}$ as initial sequents and ends with the empty sequent $\Rightarrow$. The initial sequents from $\mathcal{S}$ are called *nonlogical axioms*. Since the empty sequent is false under any truth assignment, an LK refutation proves the unsatisfiability of the nonlogical axioms $\mathcal{S}$.

As already mentioned, Frege systems are strictly stronger than resolution; in fact, even the weaker bounded-depth Frege systems discussed in the next section are more powerful: they can polynomially simulate resolution, but cannot be polynomially simulated by resolution. It was shown in [Bus87] that Frege systems have polynomial-size proofs of the pigeonhole principle formulas, whereas, as already mentioned several times above, the smallest resolution refutations grow exponentially [Hak85].

The polynomial-size Frege proofs of pigeonhole principle formulas can be constructed by showing that Frege systems can define counting, and even define the sum of a vector of integers when the integers are presented as propositional variables representing the integers in binary notation ([Bus87]. This was exploited in [Goe90] to prove that Frege systems can polynomially simulate cutting planes proofs. Furthermore, it is not hard to see that Frege systems can polynomially simulate Nullstellensatz and polynomial calculus over any finite field $\mathbb{F}$.

Extended Frege systems are conjectured to be yet stronger than Frege systems, but we lack good examples of combinatorial properties that might have short extended Frege proofs but require exponentially long Frege proofs. The paper [BBP95] identified some candidates, but these have largely been ruled out in recent years [HT15, TC17, ABB16, Bus15].

Open Problem 7.19. Find new combinatorial candidates for exponentially separating Frege and extended Frege proofs.

Open Problem 7.20. Give a conditional exponential lower bound for Frege proofs using some assumption weaker than $\mathsf{NP} \neq \mathsf{coNP}$.

7.10. Bounded-Depth Frege Proof System

One important measure of the complexity of general propositional formulas with the connectives $\wedge$ and $\vee$ is the number of alternations of $\wedge$'s and $\vee$'s. The *depth* of a propositional formula φ equals the number of levels of disjunctions and conjunctions in φ. More formally, any literal x or $\overline{x}$ is a depth-0 formula, and if $\varphi_1, \ldots, \varphi_k$ are all depth strictly smaller than d, then any formula formed by combining those k formulas with only conjunctions, or with only disjunctions, has depth bounded by d. For example, $v \wedge ((x \vee (\overline{y} \vee z)) \wedge \overline{u})$ is a depth-2 formula.

The sequent calculus LK also lends itself to working with Frege proofs with restrictions on formula depth in a natural way. At the bottom level, resolution can be viewed as a depth-0 LK refutation system. In particular, a resolution refutation can be viewed as an LK refutation in which all formulas are merely propositional

variables. To see this, note that if C is a clause containing the variables $x_1, \ldots, x_k$ unnegated and the negated literals $\overline{y}_1, \ldots, \overline{y}_{k'}$, then the corresponding sequent S_C is $y_1, \ldots, y_{k'} \Rightarrow x_1, \ldots, x_k$. Under this translation between clauses and sequents of literals, the resolution inference rule corresponds exactly to a cut inference in depth-0 LK. In this way, resolution refutations are essentially identical to depth-0 LK refutations.

The notion of depth-0 LK proof can be generalized to any bounded depth d. For $d \geq 0$, a *depth-d LK proof* is an LK proof in which every formula has depth at most d. We write *d-LK* as a compact notation for the LK proof system restricted to depth-d proofs.

Loosely speaking, the *depth-d Frege system* is the same as the d-LK proof system. However, the notion of depth for Frege proofs is sensitive to issues such as whether proofs are tree-like or DAG-like, and whether proofs are formulated Hilbert-style with modus ponens (say) or with the sequent calculus. Thus, to be formal, we will restrict our attention to the (DAG-like) d-LK proof system.

The *Res*(k) or *k-DNF resolution* proof system is an extension of resolution in which the lines are DNF formulas where the conjunctions have size bounded by k. Note that *Res*(1) is the same as resolution, so *Res*(k) proof systems lie just above resolution in strength. *Res*(k) can also be defined as the subsystem of depth-1 LK in which all formulas are either disjunctions or conjunctions of at most k literals. A number of exponential lower bounds on *Res*(k) refutations of weak pigeonhole principle formulas have been proven in a sequence of papers [ABE02, SBI04, Ale11, AMO15, Raz15] with Razborov [Raz15] establishing exponential lower bounds for $k = \epsilon \log n / \log\log n$. Size-space trade-offs for *Res*(k) were obtained in [BN11].

There are also exponential lower bounds on the size of depth-d LK proofs for all constant $d > 0$. Haken's exponential lower bound for resolution refutations of pigeonhole principle formulas [Hak85] applies to depth-0 LK. This was substantially extended to give strong lower bounds on the size of bounded-depth LK proofs of the pigeonhole principle formulas by [PBI93, KPW95] who showed that with $m = n+1$ pigeons and n holes, depth d LK refutations of the PHP formulas require size $\exp\bigl(\Omega\bigl(n^{5^{1/d}}\bigr)\bigr)$, improving on earlier lower bounds of [Ajt88].

Other known hard principles for bounded-depth LK include the ***counting-mod-p principle*** formulas.[49] We define here only the case $p = 2$; this is the ***parity principle***. Fix $n > 0$ an odd integer. The parity principle $PARITY_n$ uses variables $x_{i,j}$ for $1 \leq i < j \leq n$; for convenience of notation, we define $x_{j,i}$ to be the same variable as $x_{i,j}$. The clauses of $PARITY_n$ consist of the n many totality clauses

$$\bigvee_{j \neq i} x_{i,j} \qquad \text{for } 1 \leq i \leq n \tag{7.76a}$$

and the $n\binom{n-1}{2}$ many clauses

$$\overline{x}_{i,j} \vee \overline{x}_{i',j} \qquad \text{for } i, i', j \in [n],\ i \neq i' \neq j \neq i \tag{7.76b}$$

[49] Counting-mod-p principle formulas can be generated in the standard DIMACS format for CNF formulas used by SAT solvers by the tool *CNFgen* [LENV17, CNF] using the command line `cnfgen count` ⟨`n`⟩ ⟨`p`⟩ for positive integers n and p such that p does not divide n (in order to get unsatisfiable instances).

claiming that there is a partition of $[n]$ into pairs of elements. These parity principle formulas $PARITY_n$[50] require exponential-size bounded-depth LK refutations; it was established in [BP96a] that depth-d LK refutations of $PARITY_n$ require size $\exp\big(\Omega\big(n^{6^{1/(d+1)}}\big)\big)$. This lower bound holds even if formulas expressing the unsatisfiability of PHP_n^{n+1} are permitted as additional axioms.

The lower bounds for the parity principle refutations can also be generalized to exponential lower bounds for general ***counting-mod-p*** principles for $p > 2$, as proved by [BIK$^+$97] building on work of [BIK$^+$96] and [Rii97a, Rii97b].

There are a number of other exponential lower bounds known for bounded-depth LK systems, but these two results are representative of the best known lower bounds. A recent result of [PRST16] has opened up a potential avenue for new lower bounds, by showing a lower bound of $\exp(\Omega((\log n)^2/d^2))$ for the size of d-LK refutations of Tseitin formulas on 3-regular expander graphs where n is the number of vertices in the graph. This result implies a *depth lower bound* of $d = \Omega(\sqrt{\log n})$ for polynomial-size d-LK refutations of these Tseitin formulas. Håstad [Hås17] instead focused on Tseitin formulas defined over grid graphs to obtain an improved size lower bound for d-LK of $\exp\big(\Omega\big(n^{1/58(d+1)}\big)\big)$ and an improved depth lower bound of $\Omega(\log n/\log\log n)$.

The above-discussed results are some of the highlights regarding lower bounds on the size of bounded-depth LK proofs. Along with the lower bounds for polynomial calculus in Section 7.5.2 and cutting planes in Section 7.7.2, these are the state of the art in proving size lower bounds on proof size.

There are also a large number of open problems remaining about the size of bounded-depth LK proofs. Many of these problems lie right at the border of where we expect to be able to prove lower bounds on the complexity of proofs without actually resolving fundamental open questions such as whether $\mathsf{P} \neq \mathsf{NP}$ or not. One such open problem is to give better separations of depth-d and depth-$(d{+}1)$ systems.

Open Problem 7.21. Is there an exponential separation between the size of depth-d LK and depth-$(d+1)$ LK refutations of CNF formulas?

So far, only a superpolynomial separation is known for d-LK versus $(d{+}1)$-LK proofs when refuting CNF formulas. This separation is obtained by, on the one hand, using the above-discussed lower bounds for d-LK proofs for pigeonhole principle formulas with $n+1$ pigeons and n holes and, on the other hand, constructing small $(d{+}1)$-LK proofs of the pigeonhole principle formulas using a speedup technique of Nepomnjaščiĭ [Nep70]. Discussing this in more detail is beyond the scope of this survey; the only proof in the literature is described in [KI02] using formalizations in bounded arithmetic.

One of the reasons that Open Problem 7.21 is so tantalizing is that the Yao-Håstad switching lemmas [Hås86] in circuit complexity tell us that depth-$(d{+}1)$ propositional formulas can be exponentially more succinct in expressibility than depth-d propositional formulas. This seems like it should give $(d{+}1)$-LK proofs more power than d-LK proofs; however, Open Problem 7.21 remains unsolved even though considerable effort has been spent on it.

[50] Parity principle formulas can be generated by *CNFgen* with the command line `cnfgen parity ⟨n⟩` with the integer n chosen odd (to get unsatisfiable instances).

Another open problem concerns the power of bounded-depth LK when the language is extended to include the parity connective $\oplus$ (exclusive or) in addition to $\wedge$ and $\vee$. It is straightforward to extend LK to accommodate formulas that use the connectives $\wedge$, $\vee$ and $\oplus$; and the notion of depth extends naturally to the notion of $\{\wedge, \vee, \oplus\}$-depth. We write d-LK($\oplus$) to denote the depth-d LK system so extended. There are polynomial size, bounded-depth LK($\oplus$) proofs of many principles that depend on only counting mod 2; these include the Tseitin formulas and the onto functional pigeonhole principle formulas for $m = n + 1$ pigeons and n holes. Fundamental results of Razborov[Raz87] and Smolensky [Smo87] in circuit complexity give exponential lower bounds on the expressibility of propositional formulas using the connectives $\wedge$, $\vee$ and $\oplus$. For instance the mod-3 summation function cannot be expressed by polynomial-size, bounded-depth propositional formulas with those connectives. Thus, bounded-depth LK($\oplus$) proofs cannot reason directly about mod-3 counting. Nonetheless, it is an open problem to obtain superpolynomial lower bounds for this proof system.

Open Problem 7.22. Let LK($\oplus$) denote LK extended with a parity (exclusive or) logical connective. Are there exponential, or even superpolynomial, lower bounds on the size of bounded-depth LK($\oplus$) proofs? Can bounded-depth LK($\oplus$) be simulated by bounded-depth LK augmented with all instances of formulas expressing the unsatisfiability of the $PARITY_n$ formulas?

Open Problem 7.22 is related to the Nullstellensatz proof system discussed in Section 7.5.1. In fact, the study of proof systems based on counting principles and counting gates was the original impetus for the development of Nullstellensatz [BIK$^+$96].

It is interesting to compare the strength of bounded-depth LK proofs to the strengths of the proof systems of resolution, cutting planes, and polynomial calculus. First, as already mentioned, d-LK polynomially simulates resolution for all $d \geq 0$. However, resolution does not polynomially simulate all the systems d-LK. An example of this are the weak pigeonhole principle formulas PHP_n^{2n} expressing that there is no injection from $2n$ pigeons to n holes. These formulas are known from [PWW88, MPW02] to have quasipolynomial-size 2-LK proofs; but from [Hak85, BT88] to require exponential-size resolution refutations.

Bounded-depth LK systems are not directly comparable to cutting planes. In one direction, there are short cutting planes refutations of PHP and parity principle formulas, for instance. However, these require exponential-size d-LK proofs for any fixed $d > 0$. For the other direction, an exponential separation can be obtained using a "weak" clique-coclique principle. Recall that the clique-coclique formulas ((7.63a)–(7.63e)) encode that a graph cannot have both a clique of size $m + 1$ and a colouring of size m. By the "weak" clique-coclique formulas we mean an analogous version that states that a graph cannot have clique of size $2m$ and a colouring of size m (this is a weaker claim, and so should be easier to prove). These latter formulas have quasipolynomial-size 3-LK proofs by adapting the quasipolynomial-size 2-LK proofs for weak PHP formulas. On the other hand, the Craig interpolation method of [Pud97], along with lower bounds on monotone real circuits based on [AB87], still work for the weak clique-coclique formulas to show that they require exponential-size cutting planes refutations.

Finally, we can compare bounded-depth Frege to polynomial calculus. In one direction, there are CNF formulas that are easy to refute in polynomial calculus, or even Nullstellensatz, but require exponential-size d-LK proofs for any fixed d, for instance, mod-p counting principles when working over a field of characteristic p. A separation in the other direction follows from [GL10b]. Thus, bounded-depth Frege and polynomial calculus are incomparable proof systems.

7.11. Concluding Remarks

In this chapter we have presented an overview of proof complexity with a focus on connections with SAT solving. On the proof complexity side, one main take-away message is that resolution is fairly well-understood, although substantial important open questions still remain open. There has also been good progress in research on polynomial calculus, cutting planes and bounded-depth Frege systems, but these proof systems are not nearly as well understood as resolution.

When it comes to applied SAT solving (where, as mentioned in the introductory section, we are mostly restricted to studying unsatisfiable formulas, since this is the setting in which the tools from proof complexity can most naturally be brought to bear), there is still lots of room for improvement in our understanding of why different formulas are easy or hard, why different proof systems are better or worse for proof search, and why sometimes seemingly very small differences in algorithms can make big differences in the effectiveness of SAT solvers. It would be interesting to investigate further whether there are relevant connections between proof complexity measures and hardness of SAT, and whether proof complexity can help to shed light on the inner workings on SAT solvers.

We want to stress again that there are many topics in proof complexity that we have not been able to cover in this survey chapter due to the finiteness of time and space, and to a large extent our choice of topics has been guided by considerations as to what should be most relevant in an applied SAT solving context. One major omission from a theoretical point of view is the topic of *bounded arithmetic*. Bounded arithmetic theories are a collection of first-order and second-order theories in formal logic with close connections to both proof complexity and computational complexity theory more broadly. An overview of bounded arithmetic can be found in the books [Bus86, Kra95, CN10] and the final part of [HP93], and the connections to proof complexity are covered in [Kra95, CN10]. The *Paris-Wilkie translation* [PW85] gives a direct connection between bounded arithmetic and quasipolynomial-size, bounded-depth Frege proofs. *Cook translations* [Coo75, Ara00, CM05, CN10, BPT14] relate bounded arithmetic theories to the Frege and extended Frege proof systems.

Another important topic that we have not discussed is *symmetry reasoning* (see Chapter 7.11). Many of the combinatorial formulas that we have examined in this survey chapter are hard because they contain symmetries that make them hard to deal with for different proof systems — examples include pigeonhole principle formulas for resolution and polynomial calculus, clique-coclique formulas for cutting planes, and (perhaps less obviously) Tseitin formulas for bounded-depth Frege — but these formulas become trivial once reasoning with symmetries is allowed. In practice, different methods of handling symmetries that use static

or dynamic techniques, or combinations thereof, have been developed in, e.g., [DBBD16, DBB17, MBCK18, MBK19, TD19] and have been shown to be quite efficient in many cases, and the question of how to express symmetry reasoning as DRAT proofs has been studied in [HHW15]. It would be very interesting to perform a theoretical study of the strength of the methods of reasoning in the above-mentioned papers. However, we are not aware of any work in proof complexity that focuses on proof systems enhanced with rules for symmetry reasoning apart from a few (by now rather old) papers such as [Kri85, Urq99, AU00, Sze05].

We also want to mention briefly a (somewhat unexpected) connection between SAT solving and MaxSAT solving (discussed in Chapter 7.11). In general, it is more difficult to solve MaxSAT problems than SAT problems. However, *dual-rail MaxSAT* has recently been introduced in the papers [IMM17, BBI$^+$18, MIB$^+$19] as a method for solving SAT instances. A dual-rail proof system replaces each propositional variable x with two new propositional variables p_x and n_x, for the positive and negative values of x, representing the literals x and $\overline{x}$. An appropriate dual-rail formulation allows a SAT instance to be reduced to a MaxSAT instance which can be easier in some cases.

Concluding this chapter, the main algorithmic challenge we want to highlight is if and how one can build efficient SAT solvers based on stronger proof systems than resolution. Is it really the case that conflict-driven clause learning (CDCL), originating in the DPLL method from the early 1960s, is the best conceivable paradigm? Or could it be possible that is it now time, over 50 years later, to take the next step and build fundamentally different SAT solvers, perhaps based on some of the algebraic and/or geometric methods discussed in this survey chapter? Is the comparative lack of progress on SAT solvers using such stronger method of reasoning an indication of that there are fundamental limitations to efficient proof search being implemented within stronger proof systems? Or could it be that a sustained long-term effort will yield powerful new SAT solving paradigms, just as the immense work spent on optimizing CDCL solvers over the years have led to improvements in performance of several orders of magnitude?

Acknowledgements

We are most grateful to all our colleagues in the proof complexity and SAT communities, too numerous to list here, with whom we have had stimulating and enlightening discussions over the years. We want to give a special thanks to Paul Beame, Jan Elffers, Stephan Gocht, Massimo Lauria, Gunnar Stålmarck, and Marc Vinyals for answering questions and/or proof-reading different versions of the manuscript, helping to catch many typos and other mistakes, and giving suggestions how to improve the exposition. We are also thankful to Massimo Lauria and Marc Vinyals for providing information on how to generate different crafted proof complexity benchmark formulas using the tool *CNFgen* and finding suitable values for the parameters. Furthermore, we wish to thank Daniel Le Berre, João Marques-Silva, and Dmitry Sokolov for help with literature references, and Armin Biere, Laurent Simon, and Niklas Sörensson for providing empirical data on SAT solver performance. Last, but certainly not least, we are most indebted to Robert Robere for an amazingly thorough review of the first completed version

of the chapter, which further helped to enhance the manuscript. Any remaining errors or deficiencies are certainly the responsibility of the authors only.

Part of this work was carried out while the second author visited the Simons Institute for the Theory of Computing at UC Berkeley in association with the DIMACS/Simons Collaboration on Lower Bounds in Computational Complexity, which is conducted with support from the National Science Foundation. The first author was funded by the Simons Foundation, grant number 578919. The second author was funded by the Swedish Research Council (VR) grants 621-2012-5645 and 2016-00782 as well as by the Independent Research Fund Denmark grant 9040-00389B.

References

[AB87] B. Alon and R. Boppana. The monotone circuit complexity of Boolean functions. *Combinatorica*, 7:1–22, 1987.

[ABB16] James Aisenberg, Maria Luisa Bonet, and Sam Buss. Quasi-polynomial size Frege proofs of Frankl's theorem on the trace of finite sets. *Journal of Symbolic Logic*, 81(2):1–24, 2016.

[ABdR+18] Albert Atserias, Ilario Bonacina, Susanna F. de Rezende, Massimo Lauria, Jakob Nordström, and Alexander Razborov. Clique is hard on average for regular resolution. In *Proceedings of the 50th Annual ACM Symposium on Theory of Computing (STOC '18)*, pages 866–877, June 2018.

[ABE02] Albert Atserias, María Luisa Bonet, and Juan Luis Esteban. Lower bounds for the weak pigeonhole principle and random formulas beyond resolution. *Information and Computation*, 176(2):136–152, August 2002. Preliminary version in *ICALP '01*.

[ABLM08] Carlos Ansótegui, María Luisa Bonet, Jordi Levy, and Felip Manyà. Measuring the hardness of SAT instances. In *Proceedings of the 23rd National Conference on Artificial Intelligence (AAAI '08)*, pages 222–228, July 2008.

[ABRW02] Michael Alekhnovich, Eli Ben-Sasson, Alexander A. Razborov, and Avi Wigderson. Space complexity in propositional calculus. *SIAM Journal on Computing*, 31(4):1184–1211, April 2002. Preliminary version in *STOC '00*.

[AD08] Albert Atserias and Víctor Dalmau. A combinatorial characterization of resolution width. *Journal of Computer and System Sciences*, 74(3):323–334, May 2008. Preliminary version in *CCC '03*.

[AdRNV17] Joël Alwen, Susanna F. de Rezende, Jakob Nordström, and Marc Vinyals. Cumulative space in black-white pebbling and resolution. In *Proceedings of the 8th Innovations in Theoretical Computer Science Conference (ITCS '17)*, volume 67 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 38:1–38:21, January 2017.

[AFT11] Albert Atserias, Johannes Klaus Fichte, and Marc Thurley. Clause-learning algorithms with many restarts and bounded-width resolution. *Journal of Artificial Intelligence Research*, 40:353–373, January 2011. Preliminary version in *SAT '09*.

[AH19] Albert Atserias and Tuomas Hakoniemi. Size-degree trade-offs for Sums-of-Squares and Positivstellensatz proofs. In *Proceedings of the 34th Annual Computational Complexity Conference (CCC '19)*, volume 137 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 24:1–24:20, July 2019.

[AHI05] Michael Alekhnovich, Edward A. Hirsch, and Dmitry Itsykson. Exponential lower bounds for the running time of DPLL algorithms on satisfiable formulas. *Journal of Automated Reasoning*, 35(1–3):51–72, October 2005. Preliminary version in *ICALP '04*.

[AJPU07] Michael Alekhnovich, Jan Johannsen, Toniann Pitassi, and Alasdair Urquhart. An exponential separation between regular and general resolution. *Theory of Computing*, 3(5):81–102, May 2007. Preliminary version in *STOC '02*.

[Ajt88] Miklós Ajtai. The complexity of the pigeonhole principle. In *Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Science (FOCS '88)*, pages 346–355, October 1988.

[AKS10] Gilles Audemard, George Katsirelos, and Laurent Simon. A restriction of extended resolution for clause learning SAT solvers. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI '10)*, pages 15–20, July 2010.

[AKV04] Albert Atserias, Phokion G. Kolaitis, and Moshe Y. Vardi. Constraint propagation as a proof system. In *Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming (CP '04)*, volume 3258 of *Lecture Notes in Computer Science*, pages 77–91. Springer, 2004.

[Ale04] Michael Alekhnovich. Mutilated chessboard problem is exponentially hard for resolution. *Theoretical Computer Science*, 310(1–3):513–525, January 2004.

[Ale11] Michael Alekhnovich. Lower bounds for k-DNF resolution on random 3-CNFs. *Computational Complexity*, 20(4):597–614, December 2011. Preliminary version in *STOC '05*.

[ALN16] Albert Atserias, Massimo Lauria, and Jakob Nordström. Narrow proofs may be maximally long. *ACM Transactions on Computational Logic*, 17(3):19:1–19:30, May 2016. Preliminary version in *CCC '14*.

[AM99] Dimitris Achlioptas and Michael Molloy. Almost all graphs with $2.522n$ edges are not 3-colorable. *Electronic Journal of Combinatorics*, 6:R29:1–R29:9, July 1999.

[AM19] Albert Atserias and Moritz Müller. Automating resolution is NP-hard. In *Proceedings of the 60th Annual IEEE Symposium on Foundations of Computer Science (FOCS '19)*, pages 498–509, November 2019.

[AMO15] Albert Atserias, Moritz Müller, and Sergi Oliva. Lower bounds for DNF-refutations of a relativized weak pigeonhole principle. *Journal of Symbolic Logic*, 80(2):450–476, June 2015. Preliminary version in *CCC '13*.

[AO19] Albert Atserias and Joanna Ochremiak. Proof complexity meets algebra. *ACM Transactions on Computational Logic*, 20:1:1–1:46,

February 2019. Preliminary version in *ICALP '17*.

[AR03] Michael Alekhnovich and Alexander A. Razborov. Lower bounds for polynomial calculus: Non-binomial case. *Proceedings of the Steklov Institute of Mathematics*, 242:18–35, 2003. Available at `http://people.cs.uchicago.edu/~razborov/files/misha.pdf`. Preliminary version in *FOCS '01*.

[AR08] Michael Alekhnovich and Alexander A. Razborov. Resolution is not automatizable unless W[P] is tractable. *SIAM Journal on Computing*, 38(4):1347–1363, October 2008. Preliminary version in *FOCS '01*.

[Ara00] Toshiyasu Arai. A bounded arithmetic AID for Frege systems. *Annals of Pure and Applied Logic*, 103:155–199, 2000.

[AS09] Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern SAT solvers. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI '09)*, pages 399–404, July 2009.

[AS12] Gilles Audemard and Laurent Simon. Refining restarts strategies for SAT and UNSAT. In *Proceedings of the 18th International Conference on Principles and Practice of Constraint Programming (CP '12)*, volume 7514 of *Lecture Notes in Computer Science*, pages 118–126. Springer, October 2012.

[AU00] Noriko H. Arai and Alasdair Urquhart. Local symmetries in propositional logic. In *Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX '00)*, volume 1847 of *Lecture Notes in Computer Science*, pages 40–51. Springer, July 2000.

[BB12] María Luisa Bonet and Samuel R. Buss. An improved separation of regular resolution from pool resolution and clause learning. In *Proceedings of the 15th International Conference on Theory and Applications of Satisfiability Testing (SAT '12)*, volume 7317 of *Lecture Notes in Computer Science*, pages 44–57. Springer, June 2012.

[BBG+17] Patrick Bennett, Ilario Bonacina, Nicola Galesi, Tony Huynh, Mike Molloy, and Paul Wollan. Space proof complexity for random 3-CNFs. *Information and Computation*, 255:165–176, 2017.

[BBI16] Paul Beame, Chris Beck, and Russell Impagliazzo. Time-space tradeoffs in resolution: Superpolynomial lower bounds for superlinear space. *SIAM Journal on Computing*, 45(4):1612–1645, August 2016. Preliminary version in *STOC '12*.

[BBI+18] María Luisa Bonet, Samuel R. Buss, Alexey Ignatiev, João Marques-Silva, and António Morgado. MaxSAT resolution with the dual rail encoding. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI '18)*, pages 6565–6572, February 2018.

[BBJ14] María Luisa Bonet, Sam Buss, and Jan Johannsen. Improved separations of regular resolution from clause learning proof systems. *Journal of Artificial Intelligence Research*, 49:669–703, April 2014.

[BBP95] Maria Luisa Bonet, Samuel R. Buss, and Toniann Pitassi. Are there hard examples for Frege systems? In P. Clote and J. Rem-

mel, editors, *Feasible Mathematics II*, pages 30–56, Boston, 1995. Birkhäuser.

[BC96] Samuel R. Buss and Peter Clote. Cutting planes, connectivity and threshold logic. *Archive for Mathematical Logic*, 35:33–63, 1996.

[BCE+98] Paul Beame, Stephen A. 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, August 1998. Preliminary version in *STOC '95*.

[BCIP02] Joshua Buresh-Oppenheim, Matthew Clegg, Russell Impagliazzo, and Toniann Pitassi. Homogenization and the polynomial calculus. *Computational Complexity*, 11(3-4):91–108, 2002. Preliminary version in *ICALP '00*.

[BCMM05] Paul Beame, Joseph C. Culberson, David G. Mitchell, and Cristopher Moore. The resolution complexity of random graph k-colorability. *Discrete Applied Mathematics*, 153(1-3):25–47, December 2005.

[BD09] Michael Brickenstein and Alexander Dreyer. PolyBoRi: A framework for Gröbner-basis computations with Boolean polynomials. *Journal of Symbolic Computation*, 44(9):1326–1345, September 2009.

[BDG+09] Michael Brickenstein, Alexander Dreyer, Gert-Martin Greuel, Markus Wedler, and Oliver Wienand. New developments in the theory of Gröbner bases and applications to formal verification. *Journal of Pure and Applied Algebra*, 213(8):1612–1635, August 2009.

[BEGJ00] María Luisa Bonet, Juan Luis Esteban, Nicola Galesi, and Jan Johannsen. On the relative complexity of resolution refinements and cutting planes proof systems. *SIAM Journal on Computing*, 30(5):1462–1484, 2000. Preliminary version in *FOCS '98*.

[Ben09] Eli Ben-Sasson. Size-space tradeoffs for resolution. *SIAM Journal on Computing*, 38(6):2511–2525, May 2009. Preliminary version in *STOC '02*.

[Ber12] Christoph Berkholz. On the complexity of finding narrow proofs. In *Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS '12)*, pages 351–360, October 2012.

[Ber18] Christoph Berkholz. The relation between polynomial calculus, Sherali-Adams, and sum-of-squares proofs. In *Proceedings of the 35th Symposium on Theoretical Aspects of Computer Science (STACS '18)*, volume 96 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 11:1–11:14, February 2018.

[BF15] Armin Biere and Andreas Fröhlich. Evaluating CDCL variable scoring schemes. In *Proceedings of the 18th International Conference on Theory and Applications of Satisfiability Testing (SAT '15)*, volume 9340 of *Lecture Notes in Computer Science*, pages 405–422. Springer, September 2015.

[BF19] Armin Biere and Andreas Fröhlich. Evaluating CDCL restart schemes. In *Proceedings of Pragmatics of SAT 2015 and 2018*, volume 59 of *EPiC Series in Computing*, pages 1–17, March 2019. Avail-

able at https://easychair.org/publications/paper/RdBL.

[BFI+18] Paul Beame, Noah Fleming, Russell Impagliazzo, Antonina Kolokolova, Denis Pankratov, Toniann Pitassi, and Robert Robere. Stabbing planes. In *Proceedings of the 9th Innovations in Theoretical Computer Science Conference (ITCS '18)*, volume 94 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 10:1–10:20, January 2018.

[BG01] María Luisa Bonet and Nicola Galesi. Optimality of size-width tradeoffs for resolution. *Computational Complexity*, 10(4):261–276, December 2001. Preliminary version in *FOCS '99*.

[BG03] Eli Ben-Sasson and Nicola Galesi. Space complexity of random formulae in resolution. *Random Structures and Algorithms*, 23(1):92–109, August 2003. Preliminary version in *CCC '01*.

[BG15] Ilario Bonacina and Nicola Galesi. A framework for space complexity in algebraic proof systems. *Journal of the ACM*, 62(3):23:1–23:20, June 2015. Preliminary version in *ITCS '13*.

[BGIP01] Samuel R. Buss, Dima Grigoriev, Russell Impagliazzo, and Toniann Pitassi. Linear gaps between degrees for the polynomial calculus modulo distinct primes. *Journal of Computer and System Sciences*, 62(2):267–289, March 2001. Preliminary version in *CCC '99*.

[BGL13] Olaf Beyersdorff, Nicola Galesi, and Massimo Lauria. Parameterized complexity of DPLL search procedures. *ACM Transactions on Computational Logic*, 14(3):20:1–20:21, August 2013. Preliminary version in *SAT '11*.

[BGT14] Ilario Bonacina, Nicola Galesi, and Neil Thapen. Total space in resolution. In *Proceedings of the 55th Annual IEEE Symposium on Foundations of Computer Science (FOCS '14)*, pages 641–650, October 2014.

[BHJ08] Samuel R. Buss, Jan Hoffmann, and Jan Johannsen. Resolution trees with lemmas: Resolution refinements that characterize DLL-algorithms with clause learning. *Logical Methods in Computer Science*, 4(4:13), December 2008.

[BHP10] Paul Beame, Trinh Huynh, and Toniann Pitassi. Hardness amplification in proof complexity. In *Proceedings of the 42nd Annual ACM Symposium on Theory of Computing (STOC '10)*, pages 87–96, June 2010.

[BI99] Eli Ben-Sasson and Russell Impagliazzo. Random CNF's are hard for the polynomial calculus. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science (FOCS '99)*, pages 415–421, October 1999. Journal version in [BI10].

[BI10] Eli Ben-Sasson and Russell Impagliazzo. Random CNF's are hard for the polynomial calculus. *Computational Complexity*, 19(4):501–519, 2010. Preliminary version in *FOCS '99*.

[Bie06] Armin Biere. Tracecheck. http://fmv.jku.at/tracecheck/, 2006.

[Bie08] Armin Biere. Adaptive restart strategies for conflict driven SAT solvers. In *Proceedings of the 11th International Conference on Theory and Applications of Satisfiability Testing (SAT '08)*, volume 4996

of *Lecture Notes in Computer Science*, pages 28–33. Springer, May 2008.

[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 Annual IEEE Symposium on Foundations of Computer Science (FOCS '94)*, pages 794–806, November 1994.

[BIK+96] Paul Beame, Russell Impagliazzo, Jan Krajíček, Toniann Pitassi, and Pavel Pudlák. Lower bounds on Hilbert's Nullstellensatz and propositional proofs. *Proceedings of the London Mathematical Society*, 73(3):1–26, 1996.

[BIK+97] Samuel R. Buss, Russell Impagliazzo, Jan Krajíček, Pavel Pudlák, Alexander A. Razborov, and Jiri Sgall. Proof complexity in algebraic systems and bounded depth Frege systems with modular counting. *Computational Complexity*, 6(3):256–298, 1997.

[BIKS18] Samuel R. Buss, Dmitry Itsykson, Alexander Knop, and Dmitry Sokolov. Reordering rule makes OBDD proof systems stronger. In *Proceedings of the 33rd Annual Computational Complexity Conference (CCC '18)*, volume 102 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:24, June 2018.

[BIS07] Paul Beame, Russell Impagliazzo, and Ashish Sabharwal. The resolution complexity of independent sets and vertex covers in random graphs. *Computational Complexity*, 16(3):245–297, October 2007. Preliminary version in *CCC '01*.

[BIW04] Eli Ben-Sasson, Russell Impagliazzo, and Avi Wigderson. Near optimal separation of tree-like and general resolution. *Combinatorica*, 24(4):585–603, September 2004.

[BJ10] Eli Ben-Sasson and Jan Johannsen. Lower bounds for width-restricted clause learning on small width formulas. In *Proceedings of the 13th International Conference on Theory and Applications of Satisfiability Testing (SAT '10)*, volume 6175 of *Lecture Notes in Computer Science*, pages 16–29. Springer, July 2010.

[BK14] Samuel R. Buss and Leszek Kołodziejczyk. Small stone in pool. *Logical Methods in Computer Science*, 10(2):16:1–16:22, June 2014.

[BKS04] Paul Beame, Henry Kautz, and Ashish Sabharwal. Towards understanding and harnessing the potential of clause learning. *Journal of Artificial Intelligence Research*, 22:319–351, December 2004. Preliminary version in *IJCAI '03*.

[Bla37] Archie Blake. *Canonical Expressions in Boolean Algebra*. PhD thesis, University of Chicago, 1937.

[BLLM14] Armin Biere, Daniel Le Berre, Emmanuel Lonca, and Norbert Manthey. Detecting cardinality constraints in CNF. In *Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing (SAT '14)*, volume 8561 of *Lecture Notes in Computer Science*, pages 285–301. Springer, July 2014.

[BN08] Eli Ben-Sasson and Jakob Nordström. Short proofs may be spacious: An optimal separation of space and length in resolution. In

Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS '08), pages 709–718, October 2008.

[BN11] Eli Ben-Sasson and Jakob Nordström. Understanding space in proof complexity: Separations and trade-offs via substitutions. In *Proceedings of the 2nd Symposium on Innovations in Computer Science (ICS '11)*, pages 401–416, January 2011.

[BN20] Christoph Berkholz and Jakob Nordström. Supercritical space-width trade-offs for resolution. *SIAM Journal on Computing*, 49(1):98–118, February 2020. Preliminary version in *ICALP '16*.

[BNT13] Chris Beck, Jakob Nordström, and Bangsheng Tang. Some trade-off results for polynomial calculus. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing (STOC '13)*, pages 813–822, May 2013.

[Bon16] Ilario Bonacina. Total space in resolution is at least width squared. In *Proceedings of the 43rd International Colloquium on Automata, Languages and Programming (ICALP '16)*, volume 55 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 56:1–56:13, July 2016.

[BP96a] Paul Beame and Toniann Pitassi. An exponential separation between the parity principle and the pigeonhole principle. *Annals of Pure and Applied Logic*, 80(3):195–228, 1996.

[BP96b] Paul Beame and Toniann Pitassi. Simplified and improved resolution lower bounds. In *Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science (FOCS '96)*, pages 274–282, October 1996.

[BP98a] Paul Beame and Toniann Pitassi. Propositional proof complexity: Past, present, and future. *Bulletin of the European Association for Theoretical Computer Science*, 65:66–89, June 1998.

[BP98b] Samuel R. Buss and Toniann Pitassi. Good degree bounds on Nullstellensatz refutations of the induction principle. *Journal of Computer and System Sciences*, 2(57):162–171, October 1998. Preliminary version in *CCC '96*.

[BPR00] María Luisa Bonet, Toniann Pitassi, and Ran Raz. On interpolation and automatization for Frege systems. *SIAM Journal on Computing*, 29(6):1939–1967, 2000.

[BPT14] Arnold Beckmann, Pavel Pudlák, and Neil Thapen. Parity games and propositional proofs. *ACM Transactions on Computational Logic*, 15(2):17:1–30, 2014.

[Bry92] Randal E. Bryant. Symbolic Boolean manipulation with ordered binary-decision diagram. *ACM Computing Surveys*, 24(3):293–318, 1992.

[BS97] Roberto J. Bayardo Jr. and Robert Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI '97)*, pages 203–208, July 1997.

[BS14] Boaz Barak and David Steurer. Sum-of-squares proofs and the quest toward optimal algorithms. In *Proceedings*

of the International Congress of Mathematicians (ICM), volume IV, pages 509–533, August 2014. Available at `http://www.icm2014.org/download/Proceedings_Volume_IV.pdf`.

[BT88] Samuel R. Buss and Győrgy Turán. Resolution proofs of generalized pigeonhole principles. *Theoretical Computer Science*, 62:311–317, 1988.

[BT19] Samuel R. Buss and Neil Thapen. DRAT proofs, propagation redundancy, and extended resolution. In *Proceedings of the 22nd International Conference on Theory and Applications of Satisfiability Testing (SAT '19)*, volume 11628 of *Lecture Notes in Computer Science*, pages 71–89. Springer, July 2019.

[Bus86] Samuel R. Buss. *Bounded Arithmetic*. Bibliopolis, Naples, 1986. Revision of PhD thesis.

[Bus87] Samuel R. Buss. Polynomial size proofs of the propositional pigeonhole principle. *Journal of Symbolic Logic*, 52:916–927, 1987.

[Bus98] Samuel R. Buss. Lower bounds on Nullstellensatz proofs via designs. In *Proof Complexity and Feasible Arithmetics*, volume 39 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 59–71. American Mathematical Society, 1998. Available at `http://www.math.ucsd.edu/~sbuss/ResearchWeb/designs/`.

[Bus99] Samuel R. Buss. Propositional proof complexity: An introduction. In U. Berger and H. Schwichtenberg, editors, *Computational Logic*, pages 127–178. Springer-Verlag, Berlin, 1999.

[Bus12] Samuel R. Buss. Towards NP-P via proof complexity and proof search. *Annals of Pure and Applied Logic*, 163(9):1163–1182, 2012.

[Bus15] Samuel R. Buss. Quasipolynomial size proofs of the propositional pigeonhole principle. *Theoretical Computer Science*, 576:77–84, April 2015.

[BW01] Eli Ben-Sasson and Avi Wigderson. Short proofs are narrow—resolution made simple. *Journal of the ACM*, 48(2):149–169, March 2001. Preliminary version in *STOC '99*.

[CaD] CaDiCaL. `http://fmv.jku.at/cadical/`.

[CCT87] William Cook, Collette Rene Coullard, and György Turán. On the complexity of cutting-plane proofs. *Discrete Applied Mathematics*, 18(1):25–38, November 1987.

[CEI96] Matthew Clegg, Jeffery Edmonds, and Russell Impagliazzo. Using the Groebner basis algorithm to find proofs of unsatisfiability. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC '96)*, pages 174–183, May 1996.

[Chv73] Vašek Chvátal. Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete Mathematics*, 4(1):305–337, 1973.

[CIP09] Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. The complexity of satisfiability of small depth circuits. In *Revised Selected Papers from the 4th International Workshop on Parameterized and Exact Computation (IWPEC '09)*, volume 5917 of *Lecture Notes in Computer Science*, pages 75–85. Springer, September 2009.

[CK05] Donald Chai and Andreas Kuehlmann. A fast pseudo-Boolean constraint solver. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(3):305–317, March 2005. Preliminary version in *DAC '03*.

[CM05] Stephen A. Cook and Tsuyoshi Morioka. Quantified propositional calculus and a second-order theory for NC^1. *Archive for Mathematical Logic*, 44:711–749, 2005.

[CN10] Stephen A. Cook and Phuong Nguyen. *Logical Foundations of Proof Complexity*. Cambridge University Press, July 2010.

[CNF] CNFgen: Combinatorial benchmarks for SAT solvers. `https://massimolauria.net/cnfgen/`.

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing (STOC '71)*, pages 151–158, May 1971.

[Coo75] Stephen A. Cook. Feasibly constructive proofs and the propositional calculus (preliminary version). In *Proceedings of the 7th Annual ACM Symposium on Theory of Computing (STOC '75)*, pages 83–97, May 1975.

[Coo88] Stephen A. Cook. Short propositional formulas represent nondeterministic computations. *Information Processing Letters*, 26:269–270, 1988.

[CPL] IBM ILOG CPLEX optimization studio. `https://www.ibm.com/products/ilog-cplex-optimization-studio`.

[CR79] Stephen A. Cook and Robert A. Reckhow. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44(1):36–50, March 1979. Preliminary version in *STOC '74*.

[Cry] CryptoMiniSat. `https://github.com/msoos/cryptominisat/`.

[CS80] David A. Carlson and John E. Savage. Graph pebbling with many free pebbles can be difficult. In *Proceedings of the 12th Annual ACM Symposium on Theory of Computing (STOC '80)*, pages 326–332, 1980.

[CS82] David A. Carlson and John E. Savage. Extreme time-space tradeoffs for graphs with small space requirements. *Information Processing Letters*, 14(5):223–227, 1982.

[CS88] Vašek Chvátal and Endre Szemerédi. Many hard examples for resolution. *Journal of the ACM*, 35(4):759–768, October 1988.

[CT12] Eden Chlamtáč and Madhur Tulsiani. Convex relaxations and integrality gaps. In Miguel F. Anjos and Jean B. Lasserre, editors, *Handbook on Semidefinite, Conic and Polynomial Optimization*, pages 139–169. Springer, 2012.

[DBB17] Jo Devriendt, Bart Bogaerts, and Maurice Bruynooghe. Symmetric explanation learning: Effective dynamic symmetry handling for SAT. In *Proceedings of the 20th International Conference on Theory and Applications of Satisfiability Testing (SAT '17)*, volume 10491 of *Lecture Notes in Computer Science*, pages 83–100. Springer, August 2017.

[DBBD16] Jo Devriendt, Bart Bogaerts, Maurice Bruynooghe, and Marc De-

necker. Improved static symmetry breaking for SAT. In *Proceedings of the 19th International Conference on Theory and Applications of Satisfiability Testing (SAT '16)*, volume 9710 of *Lecture Notes in Computer Science*, pages 104–122. Springer, July 2016.

[DBM00] Olivier Dubois, Yacine Boufkhad, and Jacques Mandler. Typical random 3-SAT formulae and the satisfiability threshold. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '00)*, pages 126–127, January 2000.

[Dev20] Jo Devriendt. Watched propagation of 0-1 integer linear constraints. In *Proceedings of the 26th International Conference on Principles and Practice of Constraint Programming (CP '20)*, volume 12333 of *Lecture Notes in Computer Science*, pages 160–176. Springer, September 2020.

[DG02] Heidi E. Dixon and Matthew L. Ginsberg. Inference methods for a pseudo-Boolean satisfiability solver. In *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI '02)*, pages 635–640, July 2002.

[DGN20] Jo Devriendt, Ambros Gleixner, and Jakob Nordström. Learn to relax: Integrating 0-1 integer linear programming with pseudo-Boolean conflict-driven search. In *Proceedings of the 17th International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR '20)*, pages xxiv–xxv, September 2020.

[Dix04] Heidi E. Dixon. *Automating Pseudo-Boolean Inference within a DPLL Framework*. PhD thesis, University of Oregon, 2004. Available at `http://www.cirl.uoregon.edu/dixon/papers/dixonDissertation.pdf`.

[DLL62] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7):394–397, July 1962.

[DLMM08] Jesús A. De Loera, Jon Lee, Peter N. Malkin, and Susan Margulies. Hilbert's Nullstellensatz and an algorithm for proving combinatorial infeasibility. In *Proceedings of the 21st International Symposium on Symbolic and Algebraic Computation (ISSAC '08)*, pages 197–206, July 2008.

[DLMM11] Jesús A. De Loera, Jon Lee, Peter N. Malkin, and Susan Margulies. Computing infeasibility certificates for combinatorial problems through Hilbert's Nullstellensatz. *Journal of Symbolic Computation*, 46(11):1260–1283, November 2011.

[DLMO09] Jesús A. De Loera, Jon Lee, Susan Margulies, and Shmuel Onn. Expressing combinatorial problems by systems of polynomial equations and Hilbert's Nullstellensatz. *Combinatorics, Probability and Computing*, 18(04):551–582, July 2009.

[DMR09] Stefan S. Dantchev, Barnaby Martin, and Martin Rhodes. Tight rank lower bounds for the Sherali–Adams proof system. *Theoretical Computer Science*, 410(21–23):2054–2063, May 2009.

[DP60] Martin Davis and Hilary Putnam. A computing procedure for quan-

tification theory. *Journal of the ACM*, 7(3):201–215, 1960.

[DR01] Stefan S. Dantchev and Søren Riis. "Planar" tautologies hard for resolution. In *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science (FOCS '01)*, pages 220–229, October 2001.

[dRGN+20] Susanna F. de Rezende, Mika Göös, Jakob Nordström, Toniann Pitassi, Robert Robere, and Dmitry Sokolov. Automating algebraic proof systems is NP-hard. Technical Report TR20-064, Electronic Colloquium on Computational Complexity (ECCC), May 2020.

[dRLM+20] Susanna F. de Rezende, Massimo Lauria, Or Meir, Jakob Nordström, and Dmitry Sokolov. Manuscript in preparation, 2020.

[dRMN+20] Susanna F. de Rezende, Or Meir, Jakob Nordström, Toniann Pitassi, Robert Robere, and Marc Vinyals. Lifting with simple gadgets and applications to circuit and proof complexity. In *Proceedings of the 61st Annual IEEE Symposium on Foundations of Computer Science (FOCS '20)*, November 2020. To appear.

[dRNMR19] Susanna F. de Rezende, Jakob Nordström, Or Meir, and Robert Robere. Nullstellensatz size-degree trade-offs from reversible pebbling. In *Proceedings of the 34th Annual Computational Complexity Conference (CCC '19)*, volume 137 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 18:1–18:16, July 2019.

[dRNV16] Susanna F. de Rezende, Jakob Nordström, and Marc Vinyals. How limited interaction hinders real communication (and what it means for proof and circuit complexity). In *Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science (FOCS '16)*, pages 295–304, October 2016.

[DT20] Daniel Dadush and Samarth Tiwari. On the complexity of branching proofs. In *Proceedings of the 35th Annual Computational Complexity Conference (CCC '20)*, volume 169 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 34:1–34:35, July 2020.

[EGG+18] Jan Elffers, Jesús Giráldez-Cru, Stephan Gocht, Jakob Nordström, and Laurent Simon. Seeking practical CDCL insights from theoretical SAT benchmarks. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI '18)*, pages 1300–1308, July 2018.

[EGMN20] Jan Elffers, Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. Justifying all differences using pseudo-Boolean reasoning. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI '20)*, pages 1486–1494, February 2020.

[EGNV18] Jan Elffers, Jesús Giráldez-Cru, Jakob Nordström, and Marc Vinyals. Using combinatorial benchmarks to probe the reasoning power of pseudo-Boolean solvers. In *Proceedings of the 21st International Conference on Theory and Applications of Satisfiability Testing (SAT '18)*, volume 10929 of *Lecture Notes in Computer Science*, pages 75–93. Springer, July 2018.

[EJL+16] Jan Elffers, Jan Johannsen, Massimo Lauria, Thomas Magnard, Jakob Nordström, and Marc Vinyals. Trade-offs between time and

memory in a tighter model of CDCL SAT solvers. In *Proceedings of the 19th International Conference on Theory and Applications of Satisfiability Testing (SAT '16)*, volume 9710 of *Lecture Notes in Computer Science*, pages 160–176. Springer, July 2016.

[EN18] Jan Elffers and Jakob Nordström. Divide and conquer: Towards faster pseudo-Boolean solving. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI '18)*, pages 1291–1299, July 2018.

[EN20] Jan Elffers and Jakob Nordström. A cardinal improvement to pseudo-Boolean solving. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI '20)*, pages 1495–1503, February 2020.

[ES04] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In *6th International Conference on Theory and Applications of Satisfiability Testing (SAT '03), Selected Revised Papers*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2004.

[ES06] Niklas Eén and Niklas Sörensson. Translating pseudo-Boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2(1-4):1–26, March 2006.

[ET01] Juan Luis Esteban and Jacobo Torán. Space bounds for resolution. *Information and Computation*, 171(1):84–97, 2001. Preliminary versions of these results appeared in *STACS '99* and *CSL '99*.

[FKP19] Noah Fleming, Pravesh Kothari, and Toniann Pitassi. Semialgebraic proofs and efficient algorithm design. *Foundations and Trends in Theoretical Computer Science*, 14(1–2):1–221, December 2019.

[FLM+13] Yuval Filmus, Massimo Lauria, Mladen Mikša, Jakob Nordström, and Marc Vinyals. Towards an understanding of polynomial calculus: New separations and lower bounds (Extended abstract). In *Proceedings of the 40th International Colloquium on Automata, Languages and Programming (ICALP '13)*, volume 7965 of *Lecture Notes in Computer Science*, pages 437–448. Springer, July 2013.

[FLM+15] Yuval Filmus, Massimo Lauria, Mladen Mikša, Jakob Nordström, and Marc Vinyals. From small space to small width in resolution. *ACM Transactions on Computational Logic*, 16(4):28:1–28:15, July 2015. Preliminary version in *STACS '14*.

[FLN+15] Yuval Filmus, Massimo Lauria, Jakob Nordström, Noga Ron-Zewi, and Neil Thapen. Space complexity in polynomial calculus. *SIAM Journal on Computing*, 44(4):1119–1153, August 2015. Preliminary version in *CCC '12*.

[FPPR17] Noah Fleming, Denis Pankratov, Toniann Pitassi, and Robert Robere. Random $\Theta(\log n)$-CNFs are hard for cutting planes. In *Proceedings of the 58th Annual IEEE Symposium on Foundations of Computer Science (FOCS '17)*, pages 109–120, October 2017.

[Gen35] Gerhard Gentzen. Untersuchungen über das logische Schliessen I & II. *Mathematische Zeitschrift*, 39:176–210, 405–431, 1935. English translation in [Gen69], pp. 68-131.

[Gen69] Gerhard Gentzen. *Collected Papers of Gerhard Gentzen*. North-

Holland, 1969. Edited by M. E. Szabo.

[GGKS18] Ankit Garg, Mika Göös, Pritish Kamath, and Dmitry Sokolov. Monotone circuit lower bounds from resolution. In *Proceedings of the 50th Annual ACM Symposium on Theory of Computing (STOC '18)*, pages 902–911, June 2018.

[GHP02a] Dima Grigoriev, Edward A. Hirsch, and Dmitrii V. Pasechnik. Complexity of semialgebraic proofs. *Moscow Mathematical Journal*, 2(4):647–679, 2002. Preliminary version in *STACS '02*.

[GHP02b] Dima Grigoriev, Edward A. Hirsch, and Dmitrii V. Pasechnik. Exponential lower bound for static semi-algebraic proofs. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming (ICALP '02)*, volume 2380 of *Lecture Notes in Computer Science*, pages 257–268. Springer, July 2002.

[GKKS09] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. On the implementation of weight constraint rules in conflict-driven ASP solvers. In *Proceedings of the 25th International Conference on Logic Programming (ICLP '09)*, volume 5649 of *Lecture Notes in Computer Science*, pages 250–264. Springer, July 2009.

[GKMP20] Mika Göös, Sajin Koroth, Ian Mertz, and Toniann Pitassi. Automating cutting planes is NP-hard. In *Proceedings of the 52nd Annual ACM Symposium on Theory of Computing (STOC '20)*, pages 68–77, June 2020.

[GKRS19] 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 '19)*, volume 124 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 38:1–38:19, January 2019.

[GKT19] Nicola Galesi, Leszek Kołodziejczyk, and Neil Thapen. Polynomial calculus space and resolution width. In *Proceedings of the 60th Annual IEEE Symposium on Foundations of Computer Science (FOCS '19)*, pages 1325–1337, November 2019.

[GL10a] Nicola Galesi and Massimo Lauria. On the automatizability of polynomial calculus. *Theory of Computing Systems*, 47(2):491–506, August 2010.

[GL10b] Nicola Galesi and Massimo Lauria. Optimality of size-degree tradeoffs for polynomial calculus. *ACM Transactions on Computational Logic*, 12(1):4:1–4:22, November 2010.

[GMM+20] Stephan Gocht, Ross McBride, Ciaran McCreesh, Jakob Nordström, Patrick Prosser, and James Trimble. Certifying solvers for clique and maximum common (connected) subgraph problems. In *Proceedings of the 26th International Conference on Principles and Practice of Constraint Programming (CP '20)*, volume 12333 of *Lecture Notes in Computer Science*, pages 338–357. Springer, September 2020.

[GMN20a] Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. Subgraph isomorphism meets cutting planes: Solving with certified solutions. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI '20)*, pages 1134–1140, July 2020.

[GMN20b] Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. VeriPB: The easy way to make your combinatorial search algorithm trustworthy. Presented at the workshop *From Constraint Programming to Trustworthy AI* at the *26th International Conference on Principles and Practice of Constraint Programming (CP '20)*. Paper available at `http://www.cs.ucc.ie/~bg6/cptai/2020/papers/CPTAI_2020_paper_2.pdf`, September 2020.

[GN03] Evgueni Goldberg and Yakov Novikov. Verification of proofs of unsatisfiability for CNF formulas. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '03)*, pages 886–891, March 2003.

[GNY19] Stephan Gocht, Jakob Nordström, and Amir Yehudayoff. On division versus saturation in pseudo-Boolean solving. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI '19)*, pages 1711–1718, August 2019.

[Goc17] Stephan Gocht. Personal communication, 2017.

[Goe90] Andreas Goerdt. Cutting plane versus Frege proof systems. In *Proceedings of the 4th International Workshop on Computer Science Logic (CSL '90)*, volume 533 of *Lecture Notes in Computer Science*, pages 174–194. Springer, October 1990.

[Gom63] Ralph E. Gomory. An algorithm for integer solutions of linear programs. In R.L. Graves and P. Wolfe, editors, *Recent Advances in Mathematical Programming*, pages 269–302. McGraw-Hill, New York, 1963.

[GP18a] Joshua A. Grochow and Toniann Pitassi. Circuit complexity, proof complexity, and polynomial identity testing: The ideal proof system. *Journal of the ACM*, 65(6):37:1–37:59, November 2018. Preliminary version in *FOCS '14*.

[GP18b] Mika Göös and Toniann Pitassi. Communication lower bounds via critical block sensitivity. *SIAM Journal on Computing*, 47(5):1778–1806, October 2018. Preliminary version in *STOC '14*.

[GPT15] Nicola Galesi, Pavel Pudlák, and Neil Thapen. The space complexity of cutting planes refutations. In *Proceedings of the 30th Annual Computational Complexity Conference (CCC '15)*, volume 33 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 433–447, June 2015.

[GT78] John R. Gilbert and Robert Endre Tarjan. Variations of a pebble game on graphs. Technical Report STAN-CS-78-661, Stanford University, 1978. Available at `http://infolab.stanford.edu/TR/CS-TR-78-661.html`.

[Gur] Gurobi optimizer. `https://www.gurobi.com/`.

[GV01] Dima Grigoriev and Nicolai Vorobjov. Complexity of Null- and Positivstellensatz proofs. *Annals of Pure and Applied Logic*, 113(1–3):153–160, December 2001.

[GZ03] Jan Frisco Groote and Hans Zantema. Resolution and binary decision diagrams cannot simulate each other polynomially. *Discrete*

Applied Mathematics, 130(2):157–171, 2003.

[Hak85] Armin Haken. The intractability of resolution. *Theoretical Computer Science*, 39(2-3):297–308, August 1985.

[Hås86] Johan Håstad. Almost optimal lower bounds for small depth circuits. In *Proceedings of the 18-th Annual ACM Symposium on Theory of Computing (STOC)*, pages 6–20, 1986.

[Hås17] Johan Håstad. On small-depth frege proofs for Tseitin for grids. In *Proceedings of the 58th Annual IEEE Symposium on Foundations of Computer Science (FOCS '17)*, pages 97–108, October 2017.

[HB18] Marijn J. H. Heule and Armin Biere. What a difference a variable makes. In *Proceedings of the 24th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 18)*, volume 10806 of *Lecture Notes in Computer Science*, pages 75–92. Springer, April 2018.

[HBPV08] Philipp Hertel, Fahiem Bacchus, Toniann Pitassi, and Allen Van Gelder. Clause learning can effectively P-simulate general propositional resolution. In *Proceedings of the 23rd National Conference on Artificial Intelligence (AAAI '08)*, pages 283–290, July 2008.

[HC99] Armin Haken and Stephen A. Cook. An exponential lower bound for the size of monotone real circuits. *Journal of Computer and System Sciences*, 58(2):326–335, April 1999.

[HHW13a] Marijn J. H. Heule, Warren A. Hunt Jr., and Nathan Wetzler. Trimming while checking clausal proofs. In *Proceedings of the 13th International Conference on Formal Methods in Computer-Aided Design (FMCAD '13)*, pages 181–188, October 2013.

[HHW13b] Marijn J. H. Heule, Warren A. Hunt Jr., and Nathan Wetzler. Verifying refutations with extended resolution. In *Proceedings of the 24th International Conference on Automated Deduction (CADE-24)*, volume 7898 of *Lecture Notes in Computer Science*, pages 345–359. Springer, June 2013.

[HHW15] Marijn J. H. Heule, Warren A. Hunt Jr., and Nathan Wetzler. Expressing symmetry breaking in DRAT proofs. In *Proceedings of the 25th International Conference on Automated Deduction (CADE-25)*, volume 9195 of *Lecture Notes in Computer Science*, pages 591–606. Springer, August 2015.

[HJB10] Marijn Heule, Matti Järvisalo, and Armin Biere. Clause elimination procedures for CNF formulas. In *Proceedings of the 17th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR '10)*, volume 6397 of *Lecture Notes in Computer Science*, pages 357–371. Springer, October 2010.

[HKB17] Marijn J. H. Heule, Benjamin Kiesl, and Armin Biere. Short proofs without new variables. In *Proceedings of the 26th International Conference on Automated Deduction (CADE-26)*, volume 10395 of *Lecture Notes in Computer Science*, pages 130–147. Springer, August 2017.

[HKB19] Marijn J. H. Heule, Benjamin Kiesl, and Armin Biere. Encoding redundancy for satisfaction-driven clause learning. In *Proceedings of*

the 25th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 19), volume 11427 of *Lecture Notes in Computer Science*, pages 41–58. Springer, April 2019.

[HKB20] Marijn J. H. Heule, Benjamin Kiesl, and Armin Biere. Strong extension-free proof systems. *Journal of Automated Reasoning*, 64(3):533–554, 2020. Extended version of [HKB17].

[HKSB17] Marijn J. H. Heule, Benjamin Kiesl, Martina Seidl, and Armin Biere. PRuning through satisfaction. In *13th International Haifa Verification Conference (HVC '17)*, volume 10629 of *Lecture Notes in Computer Science*, pages 179–194. Springer, November 2017.

[HLW06] Shlomo Hoory, Nathan Linial, and Avi Wigderson. Expander graphs and their applications. *Bulletin of the American Mathematical Society*, 43(4):439–561, October 2006.

[HN12] Trinh Huynh and Jakob Nordström. On the virtue of succinct proofs: Amplifying communication complexity hardness to time-space trade-offs in proof complexity (Extended abstract). In *Proceedings of the 44th Annual ACM Symposium on Theory of Computing (STOC '12)*, pages 233–248, May 2012.

[Hoo88] John N. Hooker. Generalized resolution and cutting planes. *Annals of Operations Research*, 12(1):217–239, December 1988.

[Hoo92] John N. Hooker. Generalized resolution for 0-1 linear inequalities. *Annals of Mathematics and Artificial Intelligence*, 6(1):271–286, March 1992.

[HP93] Petr Hájek and Pavel Pudlák. *Metamathematics of First-order Arithmetic*. Perspectives in Mathematical Logic. Springer-Verlag, Berlin, 1993.

[HP17] Pavel Hrubeš and Pavel Pudlák. Random formulas, monotone circuits, and interpolation. In *Proceedings of the 58th Annual IEEE Symposium on Foundations of Computer Science (FOCS '17)*, pages 121–131, October 2017.

[HPV77] John Hopcroft, Wolfgang Paul, and Leslie Valiant. On time versus space. *Journal of the ACM*, 24(2):332–337, April 1977. Preliminary version in *FOCS '75*.

[HS09] Hyojung Han and Fabio Somenzi. On-the-fly clause improvement. In *Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing (SAT '09)*, volume 5584 of *Lecture Notes in Computer Science*, pages 209–222. Springer, July 2009.

[HT15] Pavel Hrubeš and Iddo Tzameret. Short proofs for the determinant identities. *SIAM Journal on Computing*, 44(2):340–383, April 2015. Preliminary version in *STOC '12*.

[Hua07] Jinbo Huang. The effect of restarts on the efficiency of clause learning. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI '07)*, pages 2318–2323, January 2007.

[Hua10] Jinbo Huang. Extended clause learning. *Artificial Intelligence*, 174(15):1277–1284, October 2010.

[HvM05] Marijn J. H. Heule and Hans van Maaren. Aligning CNF- and

equivalence-reasoning. In *7th International Conference on Theory and Applications of Satisfiability Testing (SAT '04), Selected Revised Papers*, volume 3542 of *Lecture Notes in Computer Science*, pages 145–156. Springer, 2005.

[HvM06] Marijn J.H. Heule and Hans van Maaren. March_dl: Adding adaptive heuristics and a new branching strategy. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:47–59, 2006.

[IKRS17] Dmitry Itsykson, Alexander Knop, Andrei E. Romashchenko, and Dmitry Sokolov. On OBDD-based algorithms and proof systems that dynamically change order of variables. In *Proceedings of the 34th Symposium on Theoretical Aspects of Computer Science (STACS '17)*, volume 66 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 43:1–43:14, March 2017.

[IMM17] Alexey Ignatiev, António Morgado, and João Marques-Silva. On tackling the limits of resolution in SAT solving. In *Proceedings of the 20th International Conference on Theory and Applications of Satisfiability Testing (SAT '17)*, volume 10491 of *Lecture Notes in Computer Science*, pages 164–183. Springer, August 2017.

[IP01] Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, March 2001. Preliminary version in *CCC '99*.

[IPS99] Russell Impagliazzo, Pavel Pudlák, and Jiří Sgall. Lower bounds for the polynomial calculus and the Gröbner basis algorithm. *Computational Complexity*, 8(2):127–144, 1999.

[Jär11] Matti Järvisalo. On the relative efficiency of DPLL and OBDDs with axiom and join. In *Proceedings of the 17th International Conference on Principles and Practice of Constraint Programming (CP '11)*, volume 6876 of *Lecture Notes in Computer Science*, pages 429–437. Springer, September 2011.

[JdM13] Dejan Jovanovic and Leonardo de Moura. Cutting to the chase solving linear integer arithmetic. *Journal of Automated Reasoning*, 51(1):79–108, June 2013. Preliminary version in *CADE-23* 2011.

[JHB12] Matti Järvisalo, Marijn J. H. Heule, and Armin Biere. Inprocessing rules. In *Proceedings of the 6th International Joint Conference on Automated Reasoning (IJCAR '12)*, volume 7364 of *Lecture Notes in Computer Science*, pages 355–370. Springer, June 2012.

[JMNŽ12] Matti Järvisalo, Arie Matsliah, Jakob Nordström, and Stanislav Živný. Relating proof complexity measures and practical hardness of SAT. In *Proceedings of the 18th International Conference on Principles and Practice of Constraint Programming (CP '12)*, volume 7514 of *Lecture Notes in Computer Science*, pages 316–331. Springer, October 2012.

[KBK19] Daniela Kaufmann, Armin Biere, and Manuel Kauers. Incremental column-wise verifiation of arithmetic circuits using computer algebra. *Formal Methods in Systems Design*, February 2019.

[KI02] Jan Krajíček and Russell Impagliazzo. A note on conservativity relations among bounded arithmetic theories. *Mathematical Logic*

Quarterly, 48(3):375–377, 2002.

[KI06] Arist Kojevnikov and Dmitry Itsykson. Lower bounds of static Lovász–Schrijver calculus proofs for Tseitin tautologies. In *Proceedings of the 33rd International Colloquium on Automata, Languages and Programming (ICALP '06)*, volume 4051 of *Lecture Notes in Computer Science*, pages 323–334. Springer, July 2006.

[KN97] Eyal Kushilevitz and Noam Nisan. *Communication Complexity*. Cambridge University Press, 1997.

[KN20] Janne I. Kokkala and Jakob Nordström. Using resolution proofs to analyse CDCL solvers. In *Proceedings of the 26th International Conference on Principles and Practice of Constraint Programming (CP '20)*, volume 12333 of *Lecture Notes in Computer Science*, pages 427–444. Springer, September 2020.

[KPW95] Jan Krajíček, Pavel Pudlák, and Alan R. Woods. An exponential lower bound to the size of bounded depth Frege proofs of the pigeonhole principle. *Random Structures and Algorithms*, 7(1):15–40, 1995. Preliminary version in *STOC '92*.

[Kra94] Jan Krajíček. Lower bounds to the size of constant-depth propositional proofs. *Journal of Symbolic Logic*, 59:73–86, 1994.

[Kra95] Jan Krajíček. *Bounded Arithmetic, Propositional Logic and Complexity Theory*, volume 60 of *Encyclopedia of Mathematics and Its Applications*. Cambridge University Press, November 1995.

[Kra97] Jan Krajíček. Interpolation theorems, lower bounds for proof systems, and independence results for bounded arithmetic. *Journal of Symbolic Logic*, 62(2):457–486, June 1997.

[Kra08] Jan Krajíček. An exponential lower bound for a constraint propogation proof system based on ordered binary decision diagrams. *Journal of Symbolic Logic*, 73(1):227–237, 2008.

[Kra19] Jan Krajíček. *Proof Complexity*, volume 170 of *Encyclopedia of Mathematics and Its Applications*. Cambridge University Press, March 2019.

[KRH18] Benjamin Kiesl, Adrián Rebola-Pardo, and Marijn J. H. Heule. Extended resolution simulates DRAT. In *Proceedings of the 9th International Joint Conference on Automated Reasoning (IJCAR '18)*, volume 10900 of *Lecture Notes in Computer Science*, pages 516–531. Springer, July 2018.

[Kri85] Balakrishnan Krishnamurthy. Short proofs for tricky formulas. *Acta Informatica*, 22(3):253–275, August 1985.

[KSM11] Hadi Katebi, Karem A. Sakallah, and João P. Marques-Silva. Empirical study of the anatomy of modern SAT solvers. In *Proceedings of the 14th International Conference on Theory and Applications of Satisfiability Testing (SAT '11)*, volume 6695 of *Lecture Notes in Computer Science*, pages 343–356. Springer, June 2011.

[Las01a] Jean B. Lasserre. An explicit exact SDP relaxation for nonlinear 0-1 programs. In *Proceedings of the 8th International Conference on Integer Programming and Combinatorial Optimization (IPCO '01)*, volume 2081 of *Lecture Notes in Computer Science*, pages 293–303.

Springer, June 2001.

[Las01b] Jean B. Lasserre. Global optimization with polynomials and the problem of moments. *SIAM Journal of Optimization*, 11(3):796–817, 2001.

[Lau01] Monique Laurent. A comparison of the Sherali–Adams, Lovász–Schrijver and Lasserre relaxations for 0-1 programming. *Mathematics of Operations Research*, 28:470–496, 2001.

[LBD+20] Vincent Liew, Paul Beame, Jo Devriendt, Jan Elffers, and Jakob Nordström. Verifying properties of bit-vector multiplication using cutting planes reasoning. In *Proceedings of the 20th Conference on Formal Methods in Computer-Aided Design (FMCAD '20)*, pages 194–204, September 2020.

[LENV17] Massimo Lauria, Jan Elffers, Jakob Nordström, and Marc Vinyals. CNFgen: A generator of crafted benchmarks. In *Proceedings of the 20th International Conference on Theory and Applications of Satisfiability Testing (SAT '17)*, volume 10491 of *Lecture Notes in Computer Science*, pages 464–473. Springer, August 2017.

[Lev73] Leonid A. Levin. Universal sequential search problems. *Problemy peredachi informatsii*, 9(3):115–116, 1973. In Russian. Available at `http://mi.mathnet.ru/ppi914`.

[LGPC16] Jia Hui Liang, Vijay Ganesh, Pascal Poupart, and Krzysztof Czarnecki. Learning rate based branching heuristic for SAT solvers. In *Proceedings of the 19th International Conference on Theory and Applications of Satisfiability Testing (SAT '16)*, volume 9710 of *Lecture Notes in Computer Science*, pages 123–140. Springer, July 2016.

[Lin] Lingeling, Plingeling and Treengeling. `http://fmv.jku.at/lingeling/`.

[LLMO09] Jesús A. De Loera, Jon Lee, Susan Margulies, and Shmuel Onn. Expressing combinatorial problems by systems of polynomial equations and Hilbert's Nullstellensatz. *Combinatorics, Probability and Computing*, 18(4):551–582, July 2009.

[LM02] Inês Lynce and João P. Marques-Silva. Building state-of-the-art SAT solvers. In *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI '02)*, pages 166–170, May 2002.

[LMMW20] Daniel Le Berre, Pierre Marquis, Stefan Mengel, and Romain Wallon. On irrelevant literals in pseudo-Boolean constraint learning. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI '20)*, pages 1148–1154, July 2020.

[LMW20] Daniel Le Berre, Pierre Marquis, and Romain Wallon. On weakening strategies for PB solvers. In *Proceedings of the 23rd International Conference on Theory and Applications of Satisfiability Testing (SAT '20)*, volume 12178 of *Lecture Notes in Computer Science*, pages 322–331. Springer, July 2020.

[LN17] Massimo Lauria and Jakob Nordström. Graph colouring is hard for algorithms based on Hilbert's Nullstellensatz and Gröbner bases. In *Proceedings of the 32nd Annual Computational Complexity Conference (CCC '17)*, volume 79 of *Leibniz International Proceedings in*

Informatics (LIPIcs), pages 2:1–2:20, July 2017.

[LNSS20] Guillaume Lagarde, Jakob Nordström, Dmitry Sokolov, and Joseph Swernofsky. Trade-offs between size and degree in polynomial calculus. In *Proceedings of the 11th Innovations in Theoretical Computer Science Conference (ITCS '20)*, volume 151 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 72:1–72:16, January 2020.

[LP10] Daniel Le Berre and Anne Parrain. The Sat4j library, release 2.2. *Journal on Satisfiability, Boolean Modeling and Computation*, 7:59–64, July 2010.

[LR18] Daniel Le Berre and Pascal Rapicault. Boolean-based dependency management for the Eclipse ecosystem. *International Journal on Artificial Intelligence Tools*, 27(1):1840003:1–1840003:23, February 2018.

[LS91] László Lovász and Alexander Schrijver. Cones of matrices and set-functions and 0-1 optimization. *SIAM Journal on Optimization*, 1(2):166–190, 1991.

[LT82] Thomas Lengauer and Robert Endre Tarjan. Asymptotically tight bounds on time-space trade-offs in a pebble game. *Journal of the ACM*, 29(4):1087–1130, October 1982. Preliminary version in *STOC '79*.

[LTW18] Fu Li, Iddo Tzameret, and Zhengyu Wang. Characterizing propositional proofs as noncommutative formulas. *SIAM Journal on Computing*, 47(4):1424–1462, 2018.

[Mar06] Klas Markström. Locality and hard SAT-instances. *Journal on Satisfiability, Boolean Modeling and Computation*, 2(1-4):221–227, 2006.

[MBCK18] Hakan Metin, Souheib Baarir, Maximilien Colange, and Fabrice Kordon. CDCLSym: Introducing effective symmetry breaking in SAT solving. In *Proceedings of the 24th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 18)*, volume 10806 of *Lecture Notes in Computer Science*, pages 99–114. Springer, April 2018.

[MBK19] Hakan Metin, Souheib Baarir, and Fabrice Kordon. Composing symmetry propagation and effective symmetry breaking for SAT solving. In *11th NASA Formal Methods Symposium (NFM '19)*, volume 11460 of *Lecture Notes in Computer Science*, pages 316–332. Springer, May 2019.

[MHB13] Norbert Manthey, Marijn J. H. Heule, and Armin Biere. Automated reencoding of Boolean formulas. In *8th International Haifa Verification Conference (HVC '12), Revised Selected Papers*, volume 7857 of *Lecture Notes in Computer Science*, pages 102–117. Springer, 2013.

[MIB+19] António Morgado, Alexey Ignatiev, María Luisa Bonet, João Marques-Silva, and Samuel R. Buss. DRMaxSAT with MaxHS: First contact. In *Proceedings of the 22nd International Conference on Theory and Applications of Satisfiability Testing (SAT '19)*, volume 11628 of *Lecture Notes in Computer Science*, pages 239–249.

Springer, July 2019.

[MM06] Vasco M. Manquinho and João Marques-Silva. On using cutting planes in pseudo-Boolean optimization. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:209–219, March 2006. Preliminary version in *SAT '05*.

[MML14] Ruben Martins, Vasco M. Manquinho, and Inês Lynce. Open-WBO: A modular MaxSAT solver. In *Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing (SAT '14)*, volume 8561 of *Lecture Notes in Computer Science*, pages 438–445. Springer, July 2014.

[MMZ+01] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference (DAC '01)*, pages 530–535, June 2001.

[MN14] Mladen Mikša and Jakob Nordström. Long proofs of (seemingly) simple formulas. In *Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing (SAT '14)*, volume 8561 of *Lecture Notes in Computer Science*, pages 121–137. Springer, July 2014.

[MN15] Mladen Mikša and Jakob Nordström. A generalized method for proving polynomial calculus degree lower bounds. In *Proceedings of the 30th Annual Computational Complexity Conference (CCC '15)*, volume 33 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 467–487, June 2015.

[MPR20] Nathan Mull, Shuo Pang, and Alexander A. Razborov. On CDCL-based proof systems with the ordered decision strategy. In *Proceedings of the 23rd International Conference on Theory and Applications of Satisfiability Testing (SAT '20)*, volume 12178 of *Lecture Notes in Computer Science*, pages 149–165. Springer, July 2020.

[MPW02] Alexis Maciel, Toniann Pitassi, and Alan R. Woods. A new proof of the weak pigeonhole principle. *Journal of Computer and System Sciences*, 64(4):843–872, 2002. Preliminary version in *STOC '00*.

[MPW19] Ian Mertz, Toniann Pitassi, and Yuanhao Wei. Short proofs are hard to find. In *Proceedings of the 46th International Colloquium on Automata, Languages and Programming (ICALP '19)*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 84:1–84:16, July 2019.

[MS99] João P. Marques-Silva and Karem A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, May 1999. Preliminary version in *ICCAD '96*.

[Nep70] Valery A. Nepomnjaščiĭ. Rudimentary predicates and Turing computations. *Dokl. Akad. Nauk SSSR*, 195:282–284, 1970. English translation in *Soviet Math. Dokl.* 11 (1970) 1462–1465.

[Nes00] Yurii Nesterov. Squared functional systems and optimization problems. In H. Frenk, K. Roos, T. Terlaky, and S. Zhang, editors, *High Performance Optimization*, pages 405–440. Kluwer Academic Publisher, 2000.

[NH13] Jakob Nordström and Johan Håstad. Towards an optimal separation of space and length in resolution. *Theory of Computing*, 9:471–557, May 2013. Preliminary version in *STOC '08*.

[Nor09a] Jakob Nordström. Narrow proofs may be spacious: Separating space and width in resolution. *SIAM Journal on Computing*, 39(1):59–121, May 2009. Preliminary version in *STOC '06*.

[Nor09b] Jakob Nordström. A simplified way of proving trade-off results for resolution. *Information Processing Letters*, 109(18):1030–1035, August 2009.

[Nor12] Jakob Nordström. On the relative strength of pebbling and resolution. *ACM Transactions on Computational Logic*, 13(2):16:1–16:43, April 2012. Preliminary version in *CCC '10*.

[Nor13] Jakob Nordström. Pebble games, proof complexity and time-space trade-offs. *Logical Methods in Computer Science*, 9(3):15:1–15:63, September 2013.

[Nor15] Jakob Nordström. On the interplay between proof complexity and SAT solving. *ACM SIGLOG News*, 2(3):19–44, July 2015.

[Nor20] Jakob Nordström. New wine into old wineskins: A survey of some pebbling classics with supplemental results. Manuscript in preparation. To appear in *Foundations and Trends in Theoretical Computer Science*. Current draft version available at `http://www.csc.kth.se/~jakobn/research/`, 2020.

[NR18] Alexander Nadel and Vadim Ryvchin. Chronological backtracking. In *Proceedings of the 21st International Conference on Theory and Applications of Satisfiability Testing (SAT '18)*, volume 10929 of *Lecture Notes in Computer Science*, pages 111–121. Springer, July 2018.

[Pan19] Shuo Pang. Large clique is hard on average for resolution. Technical Report TR19-068, Electronic Colloquium on Computational Complexity (ECCC), April 2019.

[Par00] Pablo A. Parrilo. *Structured Semidefinite Programs and Semialgebraic Geometry Methods in Robustness and Optimization*. PhD thesis, California Institute of Technology, May 2000. Available at `http://resolver.caltech.edu/CaltechETD:etd-05062004-055516`.

[PBI93] Toniann Pitassi, Paul Beame, and Russell Impagliazzo. Exponential lower bounds for the pigeonhole principle. *Computational Complexity*, 3:97–140, 1993. Preliminary version in *STOC '92*.

[PD07] Knot Pipatsrisawat and Adnan Darwiche. A lightweight component caching scheme for satisfiability solvers. In *Theory and Applications of Satisfiability Testing (SAT 2007)*, Lecture Notes in Computer Science 4501, pages 294–299. Springer Verlag, 2007.

[PD11] Knot Pipatsrisawat and Adnan Darwiche. On the power of clause-learning SAT solvers as resolution engines. *Artificial Intelligence*, 175(2):512–525, February 2011. Preliminary version in *CP '09*.

[PF79] Nicholas Pippenger and Michael J. Fischer. Relations among complexity measures. *Journal of the ACM*, 26:361–381, 1979.

[Pip80] Nicholas Pippenger. Pebbling. Technical Report RC8258, IBM Wat-

son Research Center, 1980. In *Proceedings of the 5th IBM Symposium on Mathematical Foundations of Computer Science*.

[PR17] Toniann Pitassi and Robert Robere. Strongly exponential lower bounds for monotone computation. In *Proceedings of the 49th Annual ACM Symposium on Theory of Computing (STOC '17)*, pages 1246–1255, June 2017.

[PR18] Toniann Pitassi and Robert Robere. Lifting Nullstellensatz to monotone span programs over any field. In *Proceedings of the 50th Annual ACM Symposium on Theory of Computing (STOC '18)*, pages 1207–1219, June 2018.

[PRST16] Toniann Pitassi, Benjamin Rossman, Rocco Servedio, and Li-Yang Tan. Poly-logarithmic Frege depth lower bounds. In *Proceedings of the 48th Annual ACM Symposium on Theory of Computing (STOC '16)*, pages 644–657, June 2016.

[Pse16] Pseudo-Boolean competition 2016. `http://www.cril.univ-artois.fr/PB16/`, July 2016.

[PT16] Toniann Pitassi and Iddo Tzameret. Algebraic proof complexity: Progress, frontiers and challenges. *ACM SIGLOG News*, 3(3):21–43, August 2016.

[PTC77] Wolfgang J. Paul, Robert Endre Tarjan, and James R. Celoni. Space bounds for a game on graphs. *Mathematical Systems Theory*, 10:239–251, 1977.

[Pud97] Pavel Pudlák. Lower bounds for resolution and cutting plane proofs and monotone computations. *Journal of Symbolic Logic*, 62(3):981–998, September 1997.

[Pud99] Pavel Pudlák. On the complexity of propositional calculus. In S. Barry Cooper and John K. Truss, editors, *Sets and Proofs*, volume 258 of *London Mathematical Society Lecture Note Series*, pages 197–218. Cambridge University Press, 1999.

[PV05] Guoqiang Pan and Moshe Y. Vardi. Symbolic techniques in satisfiability solving. *Journal of Automated Reasoning*, 35(1–3):25–50, October 2005.

[PW85] Jeff B. Paris and Alex J. Wilkie. Counting problems in bounded arithmetic. In *Methods in Mathematical Logic, Lecture Notes in Mathematics #1130*, pages 317–340. Springer-Verlag, 1985.

[PWW88] Jeff B. Paris, Alex J. Wilkie, and A. R. Woods. Provability of the pigeonhole principle and the existence of infinitely many primes. *Journal of Symbolic Logic*, 53:1235–1244, 1988.

[Raz87] Alexander A. Razborov. Lower bounds on the size of bounded depth networks over a complete basis with logical addition. *Matematicheskie Zametki*, 41(4):598–607, 1987. English Translation in *Mathematical Notes of the Academy of Sciences of the USSR*, 41(4):333–338, 1987.

[Raz98] Alexander A. Razborov. Lower bounds for the polynomial calculus. *Computational Complexity*, 7(4):291–324, December 1998.

[Raz02] Alexander A. Razborov. Proof complexity of pigeonhole principles. In *5th International Conference on Developments in Language The-*

ory, (DLT '01), Revised Papers, volume 2295 of *Lecture Notes in Computer Science*, pages 100–116. Springer, July 2002.

[Raz03] Alexander A. Razborov. Resolution lower bounds for the weak functional pigeonhole principle. *Theoretical Computer Science*, 1(303):233–243, June 2003.

[Raz04a] Ran Raz. Resolution lower bounds for the weak pigeonhole principle. *Journal of the ACM*, 51(2):115–138, March 2004. Preliminary version in *STOC '02*.

[Raz04b] Alexander A. Razborov. Resolution lower bounds for perfect matching principles. *Journal of Computer and System Sciences*, 69(1):3–27, August 2004. Preliminary version in *CCC '02*.

[Raz15] Alexander A. Razborov. Pseudorandom generators hard for k-DNF resolution and polynomial calculus resolution. *Annals of Mathematics*, 181(2):415–472, March 2015.

[Raz16a] Alexander A. Razborov. A new kind of tradeoffs in propositional proof complexity. *Journal of the ACM*, 63(2):16:1–16:14, April 2016.

[Raz16b] Alexander A. Razborov. Proof complexity and beyond. *ACM SIGACT News*, 47(2):66–86, June 2016.

[RBK17] Daniela Ritirc, Armin Biere, and Manuel Kauers. Column-wise verification of multipliers using computer algebra. In *Proceedings of the 17th International Conference on Formal Methods in Computer-Aided Design (FMCAD '17)*, pages 23–30, October 2017.

[RBK18] Daniela Ritirc, Armin Biere, and Manuel Kauers. Improving and extending the algebraic approach for verifying gate-level multipliers. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE '18)*, pages 1556–1561, March 2018.

[Rec75] Robert A. Reckhow. *On the Lengths of Proofs in the Propositional Calculus*. PhD thesis, University of Toronto, 1975. Available at `https://www.cs.toronto.edu/~sacook/homepage/reckhow_thesis.pdf`.

[Rii93] Søren Riis. *Independence in Bounded Arithmetic*. PhD thesis, University of Oxford, 1993.

[Rii97a] Soren Riis. Count(q) does not imply count(q). *Annals of Pure and Applied Logic*, 90:1–56, 1997.

[Rii97b] Soren Riis. Count(q) versus the pigeon-hole principle. *Archive for Mathematical Logic*, 36:157–188, 1997.

[RM99] Ran Raz and Pierre McKenzie. Separation of the monotone NC hierarchy. *Combinatorica*, 19(3):403–435, March 1999. Preliminary version in *FOCS '97*.

[Rob65] John Alan Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, January 1965.

[Rob79] John Michael Robson. A new proof of the NP-completeness of satisfiability. In *Proceedings of the 2nd Australian Computer Science Conference*, pages 62–70, February 1979.

[Rob91] John Michael Robson. An $O(T \log T)$ reduction from RAM computations to satisfiability. *Theoretical Computer Science*, 82(1):141–149, May 1991.

[RPRC16] Robert Robere, Toniann Pitassi, Benjamin Rossman, and

Stephen A. Cook. Exponential lower bounds for monotone span programs. In *Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science (FOCS '16)*, pages 406–415, October 2016.

[RY20] Anup Rao and Amir Yehudayoff. *Communication Complexity and Applications*. Cambridge University Press, January 2020.

[Rya04] Lawrence Ryan. Efficient algorithms for clause-learning SAT solvers. Master's thesis, Simon Fraser University, February 2004. Available at `https://www.cs.sfu.ca/~mitchell/papers/ryan-thesis.ps`.

[SA90] Hanif D. Sherali and Warren P. Adams. A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems. *SIAM Journal on Discrete Mathematics*, 3:411–430, 1990.

[SAT] The international SAT Competitions web page. `http://www.satcompetition.org`.

[Sav98] John E. Savage. *Models of Computation: Exploring the Power of Computing*. Addison-Wesley, 1998. Available at `http://www.modelsofcomputation.org`.

[SB06] Carsten Sinz and Armin Biere. Extended resolution proofs for conjoining BDDs. In *Proceedings of the 1st International Computer Science Symposium in Russia (CSR '06)*, volume 3967 of *Lecture Notes in Computer Science*, pages 600–611. Springer, June 2006.

[SB09] Niklas Sörensson and Armin Biere. Minimizing learned clauses. In *Theory and Applications of Satisfiability Testing (SAT 2009)*, Lecture Notes in Computer Science 5584, pages 237–243. Springer Verlag, 2009.

[SBI04] Nathan Segerlind, Samuel R. Buss, and Russell Impagliazzo. A switching lemma for small restrictions and lower bounds for k-DNF resolution. *SIAM Journal on Computing*, 33(5):1171–1200, 2004. Preliminary version in *FOCS '02*.

[Sch78] Claus-Peter Schnorr. Satisfiability is quasilinear complete in NQL. *Journal of the ACM*, 25(1):136–145, January 1978.

[SCI] SCIP: Solving constraint integer programs. `http://scip.zib.de/`.

[SDNS20] Buser Say, Jo Devriendt, Jakob Nordström, and Peter Stuckey. Theoretical and experimental results for planning with learned binarized neural network transition models. In *Proceedings of the 26th International Conference on Principles and Practice of Constraint Programming (CP '20)*, volume 12333 of *Lecture Notes in Computer Science*, pages 917–934. Springer, September 2020.

[Seg07] Nathan Segerlind. The complexity of propositional proofs. *Bulletin of Symbolic Logic*, 13(4):417–481, December 2007.

[Seg08] Nathan Segerlind. On the relative efficiency of resolution-like proofs and ordered binary decision diagram proofs. In *Proceedings of the 23rd Annual IEEE Conference on Computational Complexity (CCC '08)*, pages 100–111, June 2008.

[Sho87] Naum Z. Shor. An approach to obtaining global extremums in polynomial mathematical programming problems. *Cybernet-*

ics, 23(5):695–700, 1987. Translated from *Kibernetika*, No. 5, pages 102-–106, 1987.

[Sim14] Laurent Simon. Post mortem analysis of SAT solver proofs. In *Proceedings of the 5th Pragmatics of SAT workshop*, volume 27 of *EPiC Series in Computing*, pages 26–40, July 2014. Available at `https://easychair.org/publications/paper/N3GD`.

[Smo87] Roman Smolensky. Algebraic methods in the theory of lower bounds for Boolean circuit complexity. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing (STOC '87)*, pages 77–82, 1987.

[SN15] Masahiko Sakai and Hidetomo Nabeshima. Construction of an ROBDD for a PB-constraint in band form and related techniques for PB-solvers. *IEICE Transactions on Information and Systems*, 98-D(6):1121–1127, June 2015.

[Spe10] Ivor Spence. sgen1: A generator of small but difficult satisfiability benchmarks. *Journal of Experimental Algorithmics*, 15:1.2:1–1.2:15, March 2010.

[SS06] Hossein M. Sheini and Karem A. Sakallah. Pueblo: A hybrid pseudo-Boolean SAT solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 2(1-4):165–189, March 2006. Preliminary version in *DATE '05*.

[Stå96] Gunnar Stålmarck. Short resolution proofs for a sequence of tricky formulas. *Acta Informatica*, 33(3):277–280, May 1996.

[Sze05] Stefan Szeider. The complexity of resolution with generalized symmetry rules. *Theory of Computing Systems*, 38(2):171–188, January 2005. Preliminary version in *STACS '03*.

[TC17] Iddo Tzameret and Stephen A. Cook. Uniform, integral and efficient proofs for the determinant identities. In *Proceedings of the 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS '17)*, pages 1–12, June 2017.

[TD19] Rodrigue Konan Tchinda and Clémentin Tayou Djamégni. Enhancing static symmetry breaking with dynamic symmetry handling in CDCL SAT solvers. *International Journal on Artificial Intelligence Tools*, 28(3):1950011:1–1950011:32, May 2019.

[Tha16] Neil Thapen. A trade-off between length and width in resolution. *Theory of Computing*, 12(5):1–14, August 2016.

[Tse68] Grigori Tseitin. The complexity of a deduction in the propositional predicate calculus. *Zapiski Nauchnyh Seminarov Leningradskogo Otdelenija matematicheskogo Instituta im. V. A. Steklova akademii Nauk SSSR (LOMI)*, 8:234–259, 1968. In Russian.

[TSZ10] Olga Tveretina, Carsten Sinz, and Hans Zantema. Ordered binary decision diagrams, pigeonhole formulas and beyond. *Journal on Satisfiability, Boolean Modeling and Computation*, 7(1):35–58, March 2010.

[Urq87] Alasdair Urquhart. Hard examples for resolution. *Journal of the ACM*, 34(1):209–219, January 1987.

[Urq99] Alasdair Urquhart. The symmetry rule in propositional logic. *Dis-*

crete Applied Mathematics, 96–97:177–193, October 1999.

[Urq11] Alasdair Urquhart. A near-optimal separation of regular and general resolution. *SIAM Journal on Computing*, 40(1):107–121, 2011. Preliminary version in *SAT '08*.

[Van05] Allen Van Gelder. Pool resolution and its relation to regular resolution and DPLL with clause learning. In *Proceedings of the 12th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR '05)*, volume 3835 of *Lecture Notes in Computer Science*, pages 580–594. Springer, 2005.

[Van08] Allen Van Gelder. Verifying RUP proofs of propositional unsatisfiability. In *10th International Symposium on Artificial Intelligence and Mathematics (ISAIM '08)*, 2008. Available at `http://isaim2008.unl.edu/index.php?page=proceedings`.

[VEG+18] Marc Vinyals, Jan Elffers, Jesús Giráldez-Cru, Stephan Gocht, and Jakob Nordström. In between resolution and cutting planes: A study of proof systems for pseudo-Boolean SAT solving. In *Proceedings of the 21st International Conference on Theory and Applications of Satisfiability Testing (SAT '18)*, volume 10929 of *Lecture Notes in Computer Science*, pages 292–310. Springer, July 2018.

[Ver19] VeriPB: Verifier for pseudo-Boolean proofs. `https://doi.org/10.5281/zenodo.3548581`, 2019.

[Vin20] Marc Vinyals. Hard examples for common variable decision heuristics. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI '20)*, pages 1652–1659, February 2020.

[VS10] Allen Van Gelder and Ivor Spence. Zero-one designs produce small hard SAT instances. In *Proceedings of the 13th International Conference on Theory and Applications of Satisfiability Testing (SAT '10)*, volume 6175 of *Lecture Notes in Computer Science*, pages 388–397. Springer, July 2010.

Index