FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Structural Reasoning Methods for Satisfiability Solving and Beyond

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der Technischen Wissenschaften

by

Dipl.-Ing. Benjamin Kiesl, BSc
Registration Number 1127227

to the Faculty of Informatics

at the TU Wien

Advisors: Assoc.-Univ.Prof. Dr. Martina Seidl
a.o. Univ.-Prof. Dr. Hans Tompits

The dissertation has been reviewed by:

Olaf Beyersdorff

Christoph Weidenbach

Vienna, 20th February, 2019

Benjamin Kiesl

Erklärung zur Verfassung der Arbeit

Dipl.-Ing. Benjamin Kiesl, BSc
Stuwerstraße 19/23
1020 Vienna
Austria

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 20. Februar 2019

Benjamin Kiesl

Acknowledgments

As a child, I once got lost in a giant supermarket. When I realized I couldn't find my parents anywhere, my muscles tightened and my heart started racing—*what if I'll never see my parents again*? Doing what a child does in that situation, I started to cry. And it worked. A lady showed up and asked me what's wrong. After young me had explained the situation to her, she first assured me that everything would be alright. She then held my hand and set out to find my parents. If you've never been in a similar situation, you don't know the relief I felt. Although my parents were nowhere to be seen, that lady seemed to know exactly what she was doing; and so, after an elaborate journey through the supermarket, we eventually found my parents. Thank you, Martina Seidl, for guiding me on this elaborate journey of my PhD.

Next, I want to thank Hans Tompits. Without his great lectures that I attended at TU Wien during my time as a master's student, I might not have started a PhD. From him, I not only learned a whole darn lot about scientific writing (I know, Hans, you wouldn't want me to use "a whole darn lot" here) but his insistence on mathematical rigour and extreme perfectionism made me a better researcher.

Although they are not mentioned on the title page of this thesis, there are two more people whose names would deserve to be there because they acted as my mentors throughout my PhD. The first is Armin Biere. If I listed all the positive things I could say about him, I would run out of space, so I keep it short: He provided me with many of the research ideas that form the basis of this thesis and he is a role model to me. The second is Marijn Heule. When I first met him in Linz in 2016, I had no idea I was speaking to a man who would go on to play such an important role in my life. During my two research visits at UT Austin, he was a better supervisor than anyone could ever ask for. Over the last two years, he has become one of my best friends.

Apart from my supervisors and mentors, I thank Olaf Beyersdorff and Christoph Weidenbach for agreeing to review this thesis. I realize that this takes a lot of time and effort.

I wouldn't have enjoyed my PhD nearly as much without my colleagues at TU Wien. Here, I want to particularly thank the famous Martin Suda, who was a great collaborator on our work regarding first-order logic, and Adrián Rebola-Pardo, without whom our IJCAR 2018 paper would not exist. Moreover, special thanks go to my colleagues within the doctoral college LogiCS, and in particular to my office mates Katalin Fazekas, Jens Katelaan, and Nadia Labai as well as to Marijana Lazić, Matthias Schlaipfer, and Ilina Stoilkovska for enduring me (I know it wasn't easy).

If we believe what some of the experts say, then *humanity* is a key ingredient of good writing. The incredible Lindy Aleshire helped me sound less like a machine and more like a human by making numerous suggestions that greatly improved this thesis. Moreover, she and her husband Devon made my stays in Austin unforgettable by taking me on several delightful trips and spending a wonderful time with me. I also want to thank Matt Kaufmann, a pedantic (in the most positive sense) proof reader who helped me refine several of our papers. My Austin room mates Rissa Jackson and Michael Rooney made me feel at home from the very first day. We had a great time sipping root beer floats and eating s'mores—without you, my research visits wouldn't have been so productive.

What would life as a PhD student look like without friends? I don't want to know. Besides those already mentioned, I want to thank the following friends for making the last four years so enjoyable: Alexander, Andi, Anna, Arjen, Benedikt, Gerald Berger, Gerald Wiesinger, Isabella, Kati, Lisa, Matthias, Max (yes, Stefan, you are meant here), Moali, Patrick, Petra, Rene, Robert, Roland, Tobias, Werner, and all friends from the Sparfari Club and from the IGGP.

You are right, David and Julia, your names are not on this list. But the reason is not that I don't consider you my friends, it's that I consider you part of my family. Thank you, David, Julia, Elfi, and Johann—the greatest in-laws I could wish for—for supporting me throughout the last four years. Thank you also to my brothers Alex and Jochen. While Jochen is clearly rooting for a finer football club (Rapid) than Alex (Barça), I love you both! Now, I really owe my parents Gertrud and Gerhard, for being there whenever I needed them—through easy and less easy times. I want you to know that you are truly fantastic parents.

Around thirteen years ago, I met a wonderful person. Since then, she's always been on my side. Sarah, thank you for your feedback to this thesis and for all the satisfiable moments we've experienced together.

Kurzfassung

Automatische Beweiser finden ihre Anwendungen in der künstlichen Intelligenz und in der formalen Verifikation, wo sie etwa zur Fehlerfindung in Software und Hardware oder zur Lösung komplexer mathematischer Probleme verwendet werden. In dieser Arbeit präsentieren wir Methoden des automatischen Beweisens, welche auf der syntaktischen Modifikation logischer Formeln basieren. Unsere Methoden beschäftigen sich mit Formeln der Aussagenlogik und der Prädikatenlogik sowie mit quantifizierten Boole'schen Formeln.

Im ersten Teil der Arbeit führen wir eine Reihe sogenannter *Redundanzkriterien* ein. Solche Redundanzkriterien beschreiben hinreichende Bedingungen, welche es erlauben, eine logische Formel zu modifizieren ohne dabei ihre Erfüllbarkeit zu beeinflussen. In einem weiteren Schritt verwenden wir dann einige dieser Redundanzkriterien, um neue Beweissysteme für die Aussagenlogik zu entwickeln. Wir demonstrieren mithilfe klassischer Methoden der Beweiskomplexität, dass unsere Beweissysteme nicht nur enorm ausdrucksstark sind, sondern dass sie sich auch bestens zur Automatisierung eignen. Um die Vorteile dieser Beweissysteme auszunutzen, führen wir dann eine neue Methode des automatischen Beweisens für die Aussagenlogik ein. Unsere Methode erweitert die erfolgreiche *conflict-driven clause learning* (CDCL) Methode, indem sie den Suchraum möglicher Wahrheitswerte massiv einschränkt. Wir zeigen experimentell, dass ein automatischer Beweiser mit unserer neuen Methode eine Reihe von logischen Formeln beweisen kann, welche aufgrund theoretischer Einschränkungen zu schwer für klassische CDCL-Beweiser sind.

Im zweiten Teil der Arbeit führen wir prädikatenlogische Generalisierungen für etliche Redundanzkriterien aus der Aussagenlogik ein. Viele dieser Redundanzkriterien wurden bisher erfolgreich zur automatischen Evaluierung aussagenlogischer Formeln verwendet, jedoch war nicht klar, ob sie auch für die Prädikatenlogik korrekt sind. Wir beweisen die Korrektheit unserer Generalisierungen mithilfe des Prinzips der *Implikation Modulo Resolution*. Das Prinzip der Implikation Modulo Resolution ist eine prädikatenlogische Verallgemeinerung von *quantified implied outer resolvents*, welche aus der Theorie quantifizierter Boole'scher Formeln stammen. In einem weiteren Schritt verwenden wir dann die generalisierten prädikatenlogischen Redundanzkritieren, um Techniken zur Redundanzelimination in logischen Formeln zu entwickeln. In einer detaillierten Analyse untersuchen wir die Konfluenzeigenschaften dieser Techniken und illustrieren deren praktischen Nutzen, indem wir einen neuen Präprozessor für prädikatenlogische Beweiser implementieren.

Anhand einer experimentellen Auswertung zeigen wir, dass durch die Verwendung dieses Präprozessors die Effizienz automatischer Beweiser signifikant erhöht werden kann.

Zu guter Letzt verwenden wir syntaktische Modifikationstechniken, um einen beweistheoretischen Zusammenhang zwischen zwei Kalkülen für quantifizierte Boole'sche Formeln, genannt *Long-Distance-Resolutionskalkül* und QRAT-*Kalkül*, herzustellen. Diverse Forschungsergebnisse aus den letzten Jahren belegen den großen praktischen Nutzen des Long-Distance-Resolutionskalküls. Es war allerdings bisher unklar, ob sich der Long-Distance-Resolutionskalkül durch den QRAT-Kalkül polynomiell simulieren lässt. Wir beweisen, dass eine solche Simulation tatsächlich möglich ist, indem wir eine Prozedur beschreiben, welche Beweise des Long-Distance-Resolutionskalküls in QRAT-Beweise transformiert.

Abstract

Automated-reasoning tools have various applications in artificial intelligence and formal verification, ranging from the detection of bugs in software and hardware to the solution of long-standing mathematical problems. In this thesis, we present automated-reasoning methods that modify the syntactic structure of logical formulas. In particular, we deal with formulas from propositional logic and first-order logic as well as with quantified Boolean formulas.

In the first part of the thesis, we introduce so-called *redundancy properties* that characterize cases in which formulas can be modified without affecting their satisfiability or unsatisfiability. Based on some of these redundancy properties, we then define new strong proof systems for propositional logic. As we demonstrate, these proof systems are not only highly expressive but also well-suited for automation. Harnessing their advantages, we define a satisfiability-solving paradigm that generalizes the well-known *conflict-driven clause learning* (CDCL) paradigm by pruning the search space more aggressively. In an empirical evaluation, we show that a solver based on our paradigm can solve formulas that are—due to theoretical restrictions—too hard for ordinary CDCL solvers.

In the second part of the thesis, we lift several popular redundancy properties from propositional logic to first-order logic. Many of these redundancy properties have been successfully used in satisfiability solving but it was unclear if they could be lifted to first-order logic. We lift them in a uniform way by introducing the principle of *implication modulo resolution*, which is a generalization of so-called *quantified implied outer resolvents* known from the theory of quantified Boolean formulas. Using these redundancy properties, we then define corresponding clause-elimination techniques and analyze their confluence properties in detail. To illustrate their practical usefulness, we implemented and evaluated a preprocessing tool that boosts the performance of theorem provers by eliminating blocked clauses from first-order formulas.

Finally, we show how satisfiability-preserving formula modifications can be used to clarify the relationship between two important proof systems for quantified Boolean formulas—the long-distance-resolution calculus and the QRAT proof system. Recently, it has been shown that long-distance resolution is remarkably powerful both in theory and in practical QBF solving. It was, however, unknown how long-distance resolution is related to QRAT, a proof system introduced for certifying the correctness of QBF-preprocessing techniques. We show that QRAT polynomially simulates long-distance resolution.

Contents

CHAPTER 1

Introduction

In this thesis, we introduce techniques that improve automated-reasoning methods by modifying the syntactic structure of logical formulas.

1.1 Historical Context and Motivation

An old man walks into a bar and orders a Scotch. The bartender asks him, "*Do you want water with your Scotch?*" to which the old man replies, "*I'm thirsty, not dirty!*" Not asking any further questions, the bartender prepares the Scotch—without water of course—and hands it over to the old man, who's happy he can finally quench his thirst. What just happened here? At no point did the old man state explicitly that he didn't want water with his Scotch; yet still, we knew exactly that he preferred his Scotch straight.

What happens in this old joke of Joe E. Lewis is that we *reason*: We combine existing facts to derive new information. When the old man claims he's thirsty and not dirty, he's telling us two facts. First, he only wants water if he's dirty, and second, he isn't dirty. From these two facts we then conclude that he doesn't want water.

Now consider the following textbook example: Suppose a street can only be wet if it rains, and that it hasn't rained. Is the street wet at the moment? Of course not. We draw this conclusion easily and we actually use the same pattern of reasoning as before. It becomes obvious when we write down the two examples below each other:

He only wants water if he's dirty.	He isn't dirty.	$\Rightarrow$	He doesn't want water.
The street is only wet if it rains.	It hasn't rained.	$\Rightarrow$	The street isn't wet.

Although the examples differ in their content—one is about a thirsty old man while the other is about a dry street—the pattern is the same:

$$A \textbf{ only if } B. \qquad \textbf{Not } B. \qquad \Rightarrow \qquad \textbf{Not } A.$$

We could even plug other statements into this pattern and—applying the pattern—we would still arrive at sound conclusions. So when we reason, we seem to apply patterns that are independent of the actual content. It is this observation that started our study of logic more than two thousand years ago, which has led to some of the greatest intellectual discoveries of humankind.

The story begins with Aristotle (384–322 BC). Although others had already dealt with some forms of reasoning before him, he can be considered the first to study logic systematically [HS18]. In the so-called *Organon*—a collection of his logical writings—he discussed several aspects of logic, most well-known among these is perhaps his theory of the *syllogism*: A syllogism is a logical pattern in which a conclusion is derived from two premises. Aristotle distinguished 256 different patterns and analyzed which of them lead to valid conclusions.

After Aristotle, logic was investigated by numerous scholars but it took many centuries until the works that most influenced modern logic were developed. Tracing back to Aristotle and strongly influenced by Wilhelm Leibniz (1646–1716), people realized at some point that logical reasoning could possibly be reduced to computation. The idea was to take certain statements and then perform a series of computation steps to derive conclusions from these statements. In order to make this work, a language is required that allows one to rigorously formulate logical statements. Enter George Boole (1815–1864). He invented such a language together with algebraic inference rules that can be used to show that a conclusion follows from given premises. Boole's impact on logic was so tremendous that even today the logical data types in most programming languages are named after him.

Now if you consider that logical reasoning is the daily bread of mathematicians—they usually reason logically to show that a theorem follows from a set of axioms—wouldn't it be great to have a precise language that is expressive enough so you can formulate mathematical statements and then check their correctness? That's what Gottlob Frege (1848–1925) must have thought before he came up with his *Begriffsschrift* (concept notation)—a language in which he could express complex statements and proofs beyond what was possible with Aristotle's syllogisms or even Boole's language. His goal was to formulate large parts of mathematics in his language in order to reduce mathematics to pure logic and to get rid of all vagueness. He had spent years developing his language and had thought he was close to his goal before receiving a devastating letter from a young Bertrand Russell (1872–1970).

Until that point, logic had been on the rise, starting out from the simple syllogisms and developing into a rich language created to formalize all of mathematics. But Russell's letter represents a turning point in the history of logic. In the letter, Russell showed how he could formulate a paradoxical statement, today known as *Russell's paradox*, in Frege's language. In particular, he was able to define the *set of all sets that don't contain* *themselves*. When we ask if that set contains itself, we end up with a contradiction: in case it contains itself, it doesn't contain itself; in case it doesn't contain itself, it contains itself. This rendered Frege's Begriffsschrift virtually useless for mathematics.

A natural consequence was then to restrict Frege's language in the hope of obtaining a logical language that forbids the formulation of paradoxical statements while still being expressive enough for mathematics. This eventually led to the development of a logic that was first presented by Hilbert and Ackermann [HA28] and that is still in use today—*first-order logic*. From then on, things went quicker than in the centuries before. In 1929, Kurt Gödel proved that for every theorem in first-order logic, there exists a proof whose correctness can be verified in a straightforward way [Göd29]. Moreover, it was—at least in theory (not in practice)—possible to find these proofs automatically. But what about sentences in first-order logic that are not theorems? Can they be identified automatically? As Alonzo Church [Chu36] and Alan Turing [Tur37] could show independently of each other, there cannot exist an algorithm that takes an arbitrary sentence in first-order logic and decides whether or not the sentence is a theorem. This fact is known as the *undecidability* of first-order logic.

So even first-order logic, the result of many centuries of research, suffers from severe restrictions. Moreover, already before the discoveries of Church and Turing, the idea of reducing mathematics to computation had been dealt a huge blow by Kurt Gödel's *incompleteness* results: He proved that in every formal system that is capable of expressing basic arithmetic, there exist true statements that are not provable within that system [Göd31]. This means that there cannot exist an algorithm that can prove all truths of mathematics.

Was it now finally time for logicians to go home and find another hobby? Not quite. The negative results of Gödel, Church, and Turing showed what cannot be done, but there were still a lot of things that could be done. This became even more true with the advent of the computer. Until then, when logicians had talked about computation, they had meant things you would do with pen and paper, or with simple machinery. But with the computer, reasoning problems of immense complexity had suddenly become approachable and so mathematicians and computer scientists set out to develop powerful automated-reasoning engines for several application areas: If you can use logic to reason about mathematical objects, why not use it for other tasks like proving the correctness of computer programs or reasoning about real-world knowledge?

History has shown that finding the perfect logical language is a hard task. Make your language too expressive and you end up with problems like Russell's paradox or the undecidability of first-order logic. Make it too simple and it becomes useless. With the availability of computers, finding the right level of expressivity has become even more important. Suddenly, it's not only relevant what sort of logical reasoning problems are solvable *in theory* but what problems can actually be solved by computers *in practice*.

Computational efficiency has become critical and people have understood that there might not be the one true logic but that the right choice of logic depends on the problem at hand. Because of this, there are now numerous different logics with varying degrees of expressivity. Still, a few of them stand out, and in this thesis we focus on three of them.

First, there is *first-order logic*, which—despite its drawbacks—is considered one of the most important logics out there. It is the logical language of mathematics and allows to model complex problems because of its high level of expressivity.

Then there is *propositional logic*, which can be seen as a restriction of first-order logic. Propositional logic is less expressive than first-order logic but if your reasoning problem can be compactly represented in propositional logic, chances are that a dedicated reasoning engine—a so-called *SAT solver*—can solve it much more efficiently (using less time and memory) than an automated reasoner for first-order logic. Because of this, it is used for all kinds of problems such as the verification of hardware and software, applications in cybersecurity, bioinformatics, and many more.

Finally, there are *quantified Boolean formulas* (QBFs). They can be seen as an intermediate logic between the other two: potentially more succinct than propositional logic while allowing for more efficient reasoning than first-order logic. They thus allow the compact formulation of problems whose representation in propositional logic might be complicated. On the other hand, not every problem from first-order logic can be expressed with QBFs and the reasoning seems harder than in propositional logic.

For all three logics, there are automated-reasoning engines which compete against each other in regular competitions to find the most efficient among them. And while some of these engines are already quite powerful, there are still many practical reasoning problems that are far beyond their reach. In this thesis, we present techniques that improve the performance of automated-reasoning engines, to make them more efficient and thus more useful.

Our approach modifies the syntactic structure of logical formulas: We remove certain redundant parts and add other useful parts, either before the actual reasoning (so-called *preprocessing*) or as an essential part of the reasoning itself. We thus view a logical formula as if it were a badly written textbook: The book might be hard to read in the beginning, but if we cross out unnecessary or misleading parts, and if we add useful comments, it might eventually become understandable enough so we can efficiently read through it.

1.2 Background

As already mentioned, we deal with propositional logic, quantified Boolean formulas (QBFs), and first-order logic. The latter two can both be seen as generalizations of propositional logic. A simple example for a propositional formula is the following one:

$$\underbrace{(x \vee \bar{y})}_{(1)} \wedge \underbrace{(\bar{x} \vee y)}_{(2)} .$$

Intuitively, this formula says that (1) x should be true *or* y should be false, *and* (2) x should be false *or* y should be true. If we pass this formula to an automated-reasoning engine, the engine tries to find out whether or not it can assign truth values (*true* or *false*) to the variables x and y such that the formula as a whole becomes true. With the above formula, we can check by hand that the formula can be made true—just assign *true* to both x and y (or, alternatively, assign *false* to both x and y).

In general, formulas of propositional logic are obtained by combining propositional variables (like x, y, and z above) and their negations ($\bar{x}$, $\bar{y}$, $\bar{z}$) with logical connectives such as '$\wedge$' ('*and*'), '$\vee$' ('*or*'), or '$\rightarrow$' ('*implies*'). By allowing quantification over the truth values of propositional variables, we obtain quantified Boolean formulas. We can use quantified Boolean formulas to ask, for instance, if there exists a truth assignment to the variable x such that, for every truth value of y, the above propositional formula is true:

$$\exists x \forall y.(x \vee \bar{y}) \wedge (\bar{x} \vee y).$$

In contrast to the propositional formula, this QBF cannot be true: in case x is true, we can make y false to make the formula false; in case x is false, we can make y true to make the formula false. The quantification over propositional variables allows for succinct formulations of reasoning problems but it also appears to make reasoning harder.

Finally, first-order logic is a generalization of propositional logic that—like quantified Boolean formulas—is obtained by adding quantification. However, in first-order logic we are not allowed to quantify over the truth values of propositional variables but over so-called *domain variables*. Moreover, instead of only simple propositional variables, first-order logic allows *predicates* over domain variables. For instance, in first-order logic we can use the domain variable x and the predicates H, R, and P to formulate the sentence "*All humans are rich or poor.*" as follows:

$$\forall x (H(x) \rightarrow R(x) \vee P(x)).$$

The formulas we considered so far are simple. In practice, however, we often deal with gigantic formulas that can contain millions of variables, and then things get more complicated. Already in propositional logic, if we attempt to evaluate a formula by naively trying out all possible assignments of truth values to its variables, we run into serious problems: for every formula with n variables, there are 2^n possible assignments, meaning that a simple formula with only 32 variables might require us to try out more than four billion assignments in the worst case.

More generally, the problem of deciding if a propositional formula is *satisfiable* (i.e., if it can be made true) is NP-complete [Coo71] and many scientists believe that there is no sub-exponential-time algorithm for this problem (a formalization of this belief is known as the *exponential-time hypothesis* [IP01]). Things seem even worse for quantified Boolean formulas, where the same problem is PSPACE-complete [MS72], and for first-order logic, where we have already seen that it is undecidable.

Considering the complexity of these reasoning problems, it's a surprise there exist automated-reasoning engines that work quite well in practice. One reason for this is that practical formulas, although large, often have particular structural properties that can be exploited by reasoning engines. On such formulas, a smart reasoning engine can do things that go way beyond the stupid brute-force approach. These things include the application of powerful inference techniques, the clever search through gigantic (in the case of first-order logic even infinite) search spaces, and the simplification of formulas before and during the actual reasoning.

Inference. The use of inference techniques can speed up reasoning significantly. As an example, assume you want to find an assignment of truth values to variables that makes the following propositional formula true:

$$x \wedge (\bar{x} \vee y) \wedge (\bar{y} \vee z).$$

Instead of trying out all possible truth assignments, you can immediately make x true because of the first part of the formula (x). After this, you have to make y true because otherwise the subformula ($\bar{x} \vee y$) becomes false. But this again forces you to make z true in order to make the last subformula, ($\bar{y} \vee z$), true. You end up with an assignment that makes the whole formula true—without having to naively try out various assignments—because you could *infer* truth values for all variables. Automated-reasoning engines make heavy use of such inference techniques to increase efficiency.

Search. In the example above, assigning truth values to the variables was straightforward. However, things are not always so clear. Oftentimes, reasoning engines for propositional logic and quantified Boolean formulas can choose between various assignments and the right choice is rarely clear. Because of this, they rely on clever heuristics that aim at solving a problem as quickly as possible [Kul09]. A similar situation arises when it comes to choosing a proper inference out of several options. Just blindly applying inferences can slow down the performance drastically. Because of this, the choice of the right inference is crucial. This is especially true in first-order logic [KV13].

Simplification. Logical formulas can contain a significant amount of redundant or misleading information. One reason for this is that formulas are often generated automatically by other tools that use a reasoning engine to solve a certain problem. But sometimes we simply don't have an explanation why the structure of a formula is suboptimal. In any case, the simplification of formulas can greatly improve the reasoning performance.

In this thesis, we introduce techniques that fall in all three of the above categories, but the focus is on inference and simplification. We achieve this by altering the syntactic structure of formulas. For instance, we introduce techniques that simply remove redundant formula parts. But we also introduce techniques that add new formula parts during the reasoning and thereby guide the search of a solver or enable new inferences. By modifying the structure of formulas, we thus influence several aspects of reasoning.

Our structural modifications are closely related to *proof systems.* Informally, a proof system defines the techniques that can be performed by an automated-reasoning engine. Moreover, in many cases we require reasoning engines to justify their results by producing a verifiable output that can be checked efficiently; this output is called a *proof*, and the form of a proof depends on the specific proof system a reasoning engine is based on. A proof system thus also describes the language in which a reasoning engine communicates its results. Therefore, because it defines the reasoning techniques and the output language, the underlying proof system of a reasoning engine affects both its runtime and the size of its output.

1.3 Contributions

The main contributions of this thesis are a range of techniques that improve automated-reasoning methods by modifying the structure of logical formulas.

We introduce several so-called *redundancy properties* for propositional logic that characterize cases in which the modification of a formula does not affect the output of a reasoning engine. More precisely, redundancy properties specify subformulas whose addition or removal does not change the satisfiability status of a formula—we call these subformulas *redundant clauses.* Based on the idea of adding redundant clauses to a formula, we then use these redundancy properties to introduce powerful *proof systems.* Using techniques from the field of *proof complexity*, we prove that our proof systems are stronger than the standard proof system in practical SAT solving (the so-called *resolution proof system*).

In particular, there exist several seemingly simple formulas for which a typical resolution-based SAT solver needs exponential time because of restrictions that apply to its underlying proof system. At the time of writing, there are no known formulas for which this restriction applies to our new proof systems. To harness the power of our new proof systems, we introduce a new SAT solving paradigm called *satisfaction-driven clause learning* (SDCL), which is a generalization of the popular *conflict-driven clause learning* (CDCL) paradigm [MSS99, MMZ$^+$01]. In an experimental evaluation, we show that a solver based on our new paradigm can prove the unsatisfiability of formulas that are beyond the reach of conventional SAT solvers.

After this, we focus again on redundant clauses, but then in first-order logic. There, we often have to deal with formulas that contain a considerable amount of redundant information. To speed up the proving process, the reasoning engines (called theorem provers) usually employ dedicated preprocessing methods that aim at simplifying formulas as much as possible. Many of these techniques eliminate redundant clauses from formulas in conjunctive normal form. However, there exists a wide variety of redundancy properties from the propositional world for which it was unclear if they could be lifted to first-order logic. We lift various of these redundancy properties to first-order logic in a uniform way by introducing the principle of *implication modulo resolution*, a first-order generalization of quantified implied outer resolvents as presented by Heule et al. [HSB16] in the context of quantified Boolean formulas.

Finally, we use syntactic modification techniques for quantified Boolean formulas to clarify the relationship between two important proof systems for QBF: the long-distance-resolution proof system and the QRAT proof system. The former is at the basis of many practical reasoning engines for QBF while the latter is able to express most preprocessing techniques used for QBF. We prove that the QRAT system is stronger than the long-distance-resolution system by presenting an algorithm that feasibly transforms long-distance-resolution proofs into QRAT proofs. Based on our algorithm, we implemented a tool that performs this transformation. With our tool it is possible to produce a single QRAT proof that combines the output of a QRAT-based preprocessor with that of a long-distance-resolution-based reasoning engine into a single uniform QRAT proof that certifies the correctness of the whole reasoning pipeline.

To summarize, our main contributions are as follows:

- We introduce novel redundancy properties for propositional logic.
- We present new proof systems for propositional logic that are based on the addition of redundant clauses.
- We introduce satisfaction-driven clause learning, a new SAT solving paradigm that harnesses the power of our new proof systems.
- We lift a range of popular redundancy properties from propositional logic to first-order logic in a uniform way.
- We introduce a procedure for quantified Boolean formulas that, based on the modification of formulas, transforms proofs from one proof system into another.

1.4 Publications

This thesis is based on the following publications (joint work is presented with the permission of all co-authors):

[KSTB16] Benjamin Kiesl, Martina Seidl, Hans Tompits, and Armin Biere. *Super-Blocked Clauses.* In: Proceedings of the 8th International Joint Conference on Automated Reasoning (IJCAR 2016), volume 9706 of LNCS, pages 45–61. Springer, 2016.

[KSS+17] Benjamin Kiesl, Martin Suda, Martina Seidl, Hans Tompits, and Armin Biere. *Blocked Clauses in First-Order Logic.* In: Proceedings of the 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR-21), volume 46 of EPiC Series in Computing, pages 31–48. EasyChair, 2017.

[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 LNCS, pages 130–147. Springer, 2017. **Best Paper Award**.

[KS17] Benjamin Kiesl and Martin Suda. *A Unifying Principle for Clause Elimination in First-Order Logic*. In: Proceedings of the 26th International Conference on Automated Deduction (CADE-26), volume 10395 of LNCS, pages 274–290. Springer, 2017. **Best Paper Award**.

[KSTB17] Benjamin Kiesl, Martina Seidl, Hans Tompits, and Armin Biere. *Blockedness in Propositional Logic: Are You Satisfied With Your Neighborhood?* In: Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI 2017), pages 4884–4888. ijcai.org, 2017. **Invited paper for the best sister conference paper track based on our IJCAR 2016 paper** [KSTB16].

[KHS17] Benjamin Kiesl, Marijn J.H. Heule, and Martina Seidl. *A Little Blocked Literal Goes a Long Way*. In: Proceedings of the 20th International Conference on Theory and Applications of Satisfiability Testing (SAT 2017), volume 10491 of LNCS, pages 281–297. Springer, 2017.

[HKSB17] Marijn J.H. Heule, Benjamin Kiesl, Martina Seidl, and Armin Biere. *PRuning Through Satisfaction*. In: Proceedings of the 13th Haifa Verification Conference (HVC 2017), volume 10629 of LNCS, pages 179–194. Springer, 2017. **Best Paper Award**.

[KSTB18] Benjamin Kiesl, Martina Seidl, Hans Tompits, and Armin Biere. *Local Redundancy in SAT: Generalizations of Blocked Clauses*. In: Logical Methods in Computer Science, vol. 14(4), 2018.

[HKB19a] Marijn Heule, Benjamin Kiesl, and Armin Biere. *Encoding Redundancy for Satisfaction-Driven Clause Learning*. In: Proceedings of the 25th Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2019). Accepted for Publication.

[HKB19b] Marijn Heule, Benjamin Kiesl, and Armin Biere. *Strong Extension-Free Proof Systems*. In: Journal of Automated Reasoning. Accepted for Publication.

The following publications were also written as part of the PhD but are beyond the scope of this thesis:

[HK17] Marijn J.H. Heule and Benjamin Kiesl. *The Potential of Interference-Based Proof Systems*. In: Proceedings of the 1st ARCADE Workshop (ARCADE 2017), volume 51 of EPiC Series in Computing, pages 51–54. EasyChair, 2017.

[KRH18] Benjamin Kiesl, Adrian Rebola-Pardo, and Marijn J.H. Heule. *Extended Resolution Simulates DRAT*. In: Proceedings of the 9th International Joint Conference on Automated Reasoning (IJCAR 2018), volume 10900 of LNCS, pages 516–531. Springer, 2018. **Best Paper Award**.

1.5 Overview

The rest of this thesis is structured as follows. In Chapter 2, which is based on our papers [KSTB16], [KSTB17], [KSTB18], [HKB17], and [HKB19b], we present new redundancy properties for propositional logic. Based on some of these redundancy properties, we then introduce new proof systems in Chapter 3, which is based on [HKB17] and [HKB19b]. In Chapter 4, we present satisfaction-driven clause learning, a SAT solving paradigm that harnesses the strengths of our new proof systems. Chapter 4 is based on our papers [HKSB17] and [HKB19a]. In Chapter 5, we lift various redundancy properties from propositional logic to first-order logic. We also present experimental evidence for the effectiveness of a clause-elimination technique based on such redundancy properties. Chapter 5 is based on [KSS+17] and [KS17], except for the sections on implication modulo flat resolution (Sections 5.2.1 and 5.2.2), which haven't yet been published. In Chapter 6, which is based on [KHS17], we present our simulation results for proof systems in QBF. Finally, in Chapter 7, we conclude and discuss future work.

CHAPTER 2

Redundant Clauses in Propositional Logic

In the following, we introduce redundancy properties that characterize cases in which the addition or removal of a formula part (a so-called *clause*) does not affect the satisfiability status of a propositional formula. Some of these redundancy properties form the basis of the proof systems and the satisfaction-driven clause learning paradigm which we will introduce in later chapters.

We consider propositional formulas in *conjunctive normal form* (CNF), which are defined as follows. A *literal* is either a variable x (a *positive literal*) or the negation $\bar{x}$ of a variable x (a *negative literal*). We say that positive (negative) literals are of positive (negative, respectively) *polarity*. The *complement* $\bar{l}$ of a literal l is defined as $\bar{l} = \bar{x}$ if $l = x$ and $\bar{l} = x$ if $l = \bar{x}$. For a literal l, we denote the variable of l by $var(l)$. A *clause* is a finite disjunction of the form $(l_1 \vee \cdots \vee l_n)$ where $l_1, \ldots, l_n$ are literals. We denote the empty clause by $\bot$. A clause that contains both a literal and its complement is a *tautology* . A *formula* is a finite conjunction of the form $C_1 \wedge \cdots \wedge C_m$ where $C_1, \ldots, C_m$ are clauses. For example, $(x \vee \bar{y}) \wedge (z) \wedge (\bar{x} \vee y \vee \bar{z})$ is a formula consisting of the clauses $(x \vee \bar{y})$, (z), and $(\bar{x} \vee y \vee \bar{z})$. If not stated otherwise, we assume that formulas do not contain tautologies. Clauses can be viewed as sets of literals, and formulas can be viewed as sets of clauses. For a set L of literals and a formula F, we define $F_L = \{C \in F \mid C \cap L \neq \emptyset\}$. We sometimes write F_l to denote $F_{\{l\}}$.

An *assignment* is a function from a (possibly infinite) set of variables to the truth values 1 (*true*) and 0 (*false*). An assignment is *total* with respect to a given formula if it assigns truth values to all variables occurring in the formula. We denote the domain of an assignment α by $var(\alpha)$. We often denote finite assignments by the sequences of literals they satisfy. For instance, the sequence $x\,\bar{y}$ denotes the assignment that assigns 1 to x and 0 to y. A literal l is *satisfied* by an assignment α if l is positive and $\alpha(var(l)) = 1$ or if l is negative and $\alpha(var(l)) = 0$. A literal is *falsified* by an assignment if its complement is satisfied by the assignment. A clause is satisfied by an assignment α if it contains a literal that is satisfied by α; it is falsified by α if α falsifies all its literals. Finally, a formula is satisfied by an assignment α if all its clauses are satisfied by α.

A formula is *satisfiable* if there exists an assignment that satisfies it, otherwise it is unsatisfiable. Two formulas are *logically equivalent* if they are satisfied by the same total assignments; they are *equisatisfiable* if they are either both satisfiable or both unsatisfiable. A formula F *implies* a clause C, denoted by $F \models C$, if every satisfying assignment of F satisfies C. Analogously, a formula F implies a formula G, denoted by $F \models G$, if every satisfying assignment of F satisfies G.

Given an assignment α and a clause C, we define $C|\alpha = \top$ if α satisfies C, otherwise $C|\alpha$ denotes the result of removing from C all literals that are falsified by α. For a formula F, we define $F|\alpha = \{C|\alpha \mid C \in F \text{ and } C|\alpha \neq \top\}$. Moreover, for a set L of literals, we denote by α_L the assignment obtained from α by assigning 1 to all the literals in L and assigning the same truth value as α to other variables not in $var(L)$. If L is a singleton set $\{l\}$, we sometimes write α_l instead of $\alpha_{\{l\}}$.

A *SAT solver* is a computer program that takes as input a propositional formula and decides whether or not the formula is satisfiable. Intuitively, we consider a clause to be redundant with respect to a formula if we can add it without affecting the result of a SAT solver:

Definition 1. *A clause C is* redundant *with respect to a formula F if F and $F \wedge C$ are equisatisfiable.*

Example 1. *The clause $(\bar{x} \vee \bar{y})$ is redundant with respect to the formula $(x \vee y)$ since $(x \vee y)$ and $(x \vee y) \wedge (\bar{x} \vee \bar{y})$ are equisatisfiable (although they are not logically equivalent).*

Note that this notion of redundancy differs from other well-known redundancy notions such as the one of Bachmair and Ganzinger usually employed within the context of ordered resolution [BG01]. Our notion of redundancy will form the basis for most of what follows. It can be used for designing techniques that simplify a formula by adding or removing redundant clauses. It can also be used for defining proof systems that allow the addition of certain types of redundant clauses.

Note that every satisfying assignment of $F \wedge C$ is trivially a satisfying assignment of F. To prove that C is redundant with respect to F it therefore suffices to show that the satisfiability of F implies the satisfiability of $F \wedge C$. To show this, we often first assume that there exists an assignment that satisfies F but falsifies C, and then we transform this assignment into a satisfying assignment of $F \wedge C$.

Deciding if a clause is redundant with respect to a formula is computationally hard in general. This led to the development of various efficiently decidable criteria that guarantee the redundancy of a clause. We call these criteria *redundancy properties*. One well-known redundancy property from the literature is the so-called *subsumption* criterion: it says that a clause C is subsumed in a formula F if F contains a clause D such that $D \subseteq C$. For example, if F contains the clause $(x \vee y)$, then the clause $(x \vee y \vee z)$ is subsumed in F. It can be easily shown that subsumed clauses are redundant. In fact, if C is subsumed in F, then F implies C. To formalize the notion of a redundancy property, and to compare different redundancy properties, we introduce the following definition:

Definition 2. *A* redundancy property *is a set of pairs* (F, C) *where* C *is redundant with respect to* F*. A redundancy property* P_1 *is more general than a redundancy property* P_2 *if* $\mathsf{P}_2 \subseteq \mathsf{P}_1$*, i.e., if every pair* $(F, C) \in \mathsf{P}_2$ *is also contained in* P_1*. If* $\mathsf{P}_2 \subset \mathsf{P}_1$*, then* P_1 *is* strictly more general *than* P_2*.*

Example 2. *The set* $\mathsf{S} = \{(F, C) \mid C \textit{ is subsumed in } F\}$ *is a redundancy property. The set* $\mathsf{IMP} = \{(F, C) \mid F \textit{ implies } C\}$ *is also a redundancy property. Since every subsumed clause is implied,* S *is a subset of* IMP *and thus* IMP *is more general than* S*.*

In what follows, we first introduce new redundancy properties that can be decided without considering the whole formula, by looking only at a subpart of the formula—the so-called *resolution neighborhood* of a clause (see Definition 5 on page 15). We call these redundancy properties *local*. The focus on local redundancy properties is motivated by the popular redundancy property of *blocked clauses* [Kul99], which we also discuss in detail. After this, we drop the locality restriction and use the insights gained from local redundancy properties to develop even more general *global* redundancy properties.

2.1 Locally Redundant Clauses

We first discuss the well-known redundancy property of blocked clauses. We then introduce the notion of a *local* redundancy property and provide examples of redundant clauses that are local but not blocked. After this, we derive a semantic notion of blocking that generalizes the traditional blocking notion, and we prove that this semantic blocking notion actually constitutes the most general local redundancy property. To bring this semantic notion of blocking closer to practical SAT solving, we come up with the syntax-based redundancy properties of *set-blocked clauses* and *super-blocked clauses*—both are strictly more general than traditional blocked clauses and for super-blocking we prove that it coincides with our semantic blocking notion. We then show how set-blocked clauses correspond to so-called *autarkies*, a well-known concept from the literature [MS85]. Finally, we analyze the complexity of deciding our new local redundancy properties before we move on to global redundancy properties in the next chapter.

2.1.1 Blocked Clauses

Blocked clauses were initially introduced by Oliver Kullmann as a generalization of the definition clauses that can be introduced in the proof system of *extended resolution* [Tse68] (see page 46). The blocked-clause definition is based on the notion of a *resolvent*:

Definition 3. *Given two clauses* C, D *and a literal* l *such that* $l \in C$ *and* $\bar{l} \in D$*, the clause* $C \otimes_l D = (C \setminus \{l\}) \cup (D \setminus \{\bar{l}\})$ *is the* resolvent *of* C *and* D *upon* l*.*

A blocked clause is a clause for which all resolvents upon one of its literals are tautologies [Kul99]:

Definition 4. *A clause* C *is* blocked *in a formula* F *if it contains a literal* l *such that for every clause* $D \in F_{\bar{l}}$*, the resolvent* $C \otimes_l D$ *is a tautology.*

We say that l blocks C in F, and we denote the set $\{(F, C) \mid C \text{ is blocked in } F\}$ by BC. Note that a clause can be blocked by more than one of its literals.

Example 3. *Consider the formula* $F = (\bar{x} \vee z) \wedge (\bar{y} \vee \bar{x})$ *and the clause* $(x \vee y)$*. The literal* x *does not block* $(x \vee y)$ *in* F *since the resolvent* $(x \vee y) \otimes_x (\bar{x} \vee z) = (y \vee z)$ *is not a tautology. However, the literal* y *blocks* $(x \vee y)$ *in* F *since the only clause in* $F_{\bar{y}}$ *is the clause* $(\bar{y} \vee \bar{x})$*, and the resolvent* $(x \vee y) \otimes_y (\bar{y} \vee \bar{x}) = (x \vee \bar{x})$ *is a tautology. Therefore,* $(x \vee y)$ *is blocked in* F*.*

There are several reasons for the popularity of blocked clauses. The elimination of blocked clauses improves the performance of modern SAT solvers [JBH10, MPW13]. Blocked clauses also provide the basis for *blocked-clause decomposition*, a technique that splits a formula into two parts that become solvable by blocked-clause elimination [HB13]. Blocked-clause decomposition is successfully used for gate extraction, for efficiently finding backbone variables, and for the detection of implied binary equivalences [BFHB14, IMS15]. Moreover, the winner of the SAT-Race 2015 competition, the solver abcdSAT [Che15], uses blocked-clause decomposition as a core technology. All this has to do with the fact that blocked clauses are redundant, which has been shown by Kullmann [Kul99]. We present the proof here because the idea behind the redundancy of blocked clauses is crucial for our later observations:

Theorem 1. BC *is a redundancy property.*

Proof. We have to show that whenever a clause is blocked in a formula, it is redundant with respect to that formula. Let C be a clause that is blocked by a literal l in a formula F and suppose there exists an assignment α that satisfies F but falsifies C. We can then easily turn α into a satisfying assignment α_l of C by flipping the truth value of l. This could only possibly falsify some of the clauses in $F_{\bar{l}}$, but the condition that l blocks C guarantees that these clauses stay satisfied: Let $D \in F_{\bar{l}}$ be such a clause. Then, since the resolvent $C \otimes_l D$ is a tautology, D must contain a literal $k \neq l$ such that $\bar{k} \in C \setminus \{l\}$. But then, since α falsifies C, it must satisfy k, and since α_l agrees with α on all literals but l, α_l satisfies D. Hence, α_l is a satisfying assignment of $F \wedge C$. We conclude that C is redundant with respect to F and thus BC is a redundancy property. □

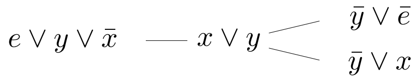


Figure 2.1: The clause $(x \vee y)$ from Example 5 and its resolution neighborhood.

In the above proof, we turn a satisfying assignment of F into a satisfying assignment of $F \wedge C$ by flipping the truth value of a single literal. This approach is used in practice to obtain a satisfying assignment of the original formula when blocked clauses have been removed during preprocessing or inprocessing [JHB12]: Suppose a SAT solver gets an input formula F and removes blocked clauses to obtain a simplified formula G. The solver then proceeds by searching for a satisfying assignment of G. Once it has found such an assignment, it can easily turn it into a satisfying assignment of the original formula F. The following example illustrates this on a concrete formula:

Example 4. *Consider again the formula* $F = (\bar{x} \vee z) \wedge (\bar{y} \vee \bar{x})$ *and the clause* $C = (x \vee y)$ *from Example 3. We already know that* y *blocks* C *in* F*. Now consider the assignment* $\alpha = \bar{x}\bar{y}\bar{z}$*, which satisfies* F *but falsifies* C*. Then, the assignment* $\alpha_y = \bar{x}y\bar{z}$*, obtained from* α *by flipping the truth value of* y*, satisfies not only* C *but also all clauses of* F*: The only clause that could have been falsified by flipping the truth value of* y *is* $(\bar{y} \vee \bar{x})$*. But, since* α *satisfies* $\bar{x}$ *and since* α_y *agrees with* α *on all variables except* y*,* α_y *satisfies* $(\bar{y} \vee \bar{x})$ *and therefore it is a satisfying assignment of* $F \wedge C$*.*

One of the particularly important properties of blocked clauses is that for testing if some clause C is blocked in a formula F it suffices to consider only those clauses of F that can be resolved with C. We call these clauses the *resolution neighborhood* of C (although our definition of the resolution neighborhood appears very natural, we are not aware that it is used elsewhere in the literature):

Definition 5. *The* resolution neighborhood $\mathsf{RN}_F(C)$ *of a clause* C *with respect to a formula* F *is the clause set* $\{D \in F \mid \exists\, l \in D \text{ such that } \bar{l} \in C\}$*.*

This raises the question if there exist redundant clauses that are not blocked but whose redundancy can be identified by considering only their resolution neighborhood. As we show in the next example, this is indeed the case:

Example 5. *Let* $C = (x \vee y)$ *and let* F *be a formula in which* C *has the resolution neighborhood* $\mathsf{RN}_F(C) = \{(e \vee y \vee \bar{x}), (\bar{y} \vee \bar{e}), (\bar{y} \vee x)\}$ *(cf. Figure 2.1). The clause* C *is not blocked in* F *but it is redundant:*

Suppose there exists an assignment α *that satisfies* F *but falsifies* C*. As we will see, we can turn* α *into a satisfying assignment of* $F \wedge C$ *by flipping the truth values of literals in* C*. By doing so, we do not affect clauses outside the resolution neighborhood of* C*. First, note that* α *must falsify both* x *and* y*, and that it must either satisfy or falsify* e*.*

In case α satisfies e, C can be satisfied by flipping the truth value of x. The only clause that could possibly be falsified by this is the clause $(e \lor y \lor \bar{x})$, but since the resulting assignment still satisfies e, the clause stays satisfied.

In case α falsifies e, we can turn it into a satisfying assignment of C by flipping the truth values of both x and y. By flipping the truth values of x and y, we could possibly falsify any of the clauses in F. But this is not the case: Since the resulting assignment satisfies y, the clause $(e \lor y \lor \bar{x})$ stays satisfied; since it falsifies e, the clause $(\bar{y} \lor \bar{e})$ stays satisfied; and since it satisfies x, the clause $(\bar{y} \lor x)$ stays satisfied.

2.1.2 A Semantic Notion of Blocking

In the examples of the preceding section, when arguing that a clause C is redundant with respect to some formula F, we showed that every assignment α that satisfies F but falsifies C can be turned into a satisfying assignment of $F \land C$ by flipping the truth values of certain literals in C. Since this flipping only affects the truth of clauses in the resolution neighborhood $\mathsf{RN}_F(C)$, it suffices to make sure that the resulting assignment satisfies $\mathsf{RN}_F(C)$ in order to guarantee that it satisfies $F \land C$. This naturally leads to the following semantic notion of blocking:

Definition 6. *A clause C is* semantically blocked *in a formula F if, for every satisfying assignment α of $\mathsf{RN}_F(C)$, there exists a set $L \subseteq C$ of literals such that α_L satisfies $\mathsf{RN}_F(C) \cup \{C\}$.*

We denote the set $\{(F, C) \mid C$ is semantically blocked in $F\}$ by $\mathsf{SEM_{BC}}$. Note that the set L of literals can possibly be empty and that a clause is semantically blocked if its resolution neighborhood is unsatisfiable. Note also that the clause C from Example 5 is semantically blocked.

Theorem 2. $\mathsf{SEM_{BC}}$ *is a redundancy property.*

Proof. Let F be a formula and let C be a clause that is semantically blocked in F. We show that $F \land C$ is satisfiable if F is satisfiable. Suppose there exists an assignment α that satisfies F but falsifies C. Since α satisfies F, it must satisfy $\mathsf{RN}_F(C)$, and since C is semantically blocked in F, there exists a set $L \subseteq C$ of literals such that α_L satisfies $\mathsf{RN}_F(C) \cup \{C\}$. Now, since α_L differs from α only on variables in $var(C)$, the only clauses in F that could possibly be falsified by α_L are those with a literal $\bar{l}$ such that $l \in C$. But those are exactly the clauses in $\mathsf{RN}_F(C)$, so α_L satisfies $F \land C$. Hence, C is redundant with respect to F and thus $\mathsf{SEM_{BC}}$ is a redundancy property. □

If a clause C is redundant with respect to some formula F and if this redundancy can be identified by considering only its resolution neighborhood in F, we expect C to be redundant with respect to every formula G in which C has the same resolution neighborhood as in F. This leads us to the notion of a *local* redundancy property:

Definition 7. *A redundancy property* P *is* local *if, for every clause* C *and any two formulas* F, G *with* $\mathsf{RN}_F(C) = \mathsf{RN}_G(C)$*, it holds that* $(F, C) \in \mathsf{P}$ *if and only if* $(G, C) \in \mathsf{P}$.

Note that this is not the only meaningful notion of locality. It would, for instance, also be possible to define locality by replacing the resolution neighborhood RN in our definition by the set of clauses whose variables coincide with the variables in C. Nevertheless, Definition 7 is the locality notion we shall use in the following. Clearly, the redundancy property of semantically blocked clauses is local:

Theorem 3. $\mathsf{SEM}_{\mathsf{BC}}$ *is a local redundancy property.*

Preparatory for showing that $\mathsf{SEM}_{\mathsf{BC}}$ is actually the most general local redundancy property (cf. Theorem 5 below), we first prove the following lemma:

Lemma 4. *If a clause* C *is not semantically blocked in a formula* F*, then there exists a formula* G *such that* $\mathsf{RN}_G(C) = \mathsf{RN}_F(C)$ *and* C *is not redundant with respect to* G.

Proof. Let F be a formula and let C be a clause that is not semantically blocked in F. This means that there exists a satisfying assignment α of $\mathsf{RN}_F(C)$ but there does not exist a set $L \subseteq C$ of literals such that α_L satisfies $\mathsf{RN}_F(C) \cup \{C\}$. In other words, there exists no satisfying assignment of $\mathsf{RN}_F(C) \cup \{C\}$ that agrees with α on all variables $x \notin var(C)$. We define a set A of unit clauses as follows:

$$A = \{(x) \mid x \notin var(C) \text{ and } \alpha(x) = 1\} \cup \{(\bar{x}) \mid x \notin var(C) \text{ and } \alpha(x) = 0\}.$$

We further define the formula $G = \mathsf{RN}_F(C) \cup A$.

Since the clauses in A contain only literals with variables that do not occur in C, it holds that neither C nor any of the clauses in A contain a literal $\bar{l}$ with $l \in C$. It therefore holds that $\mathsf{RN}_G(C) = \mathsf{RN}_F(C)$.

Now observe the following: The assignment α satisfies $\mathsf{RN}_F(C)$ and, by construction, also A, hence G is satisfiable. Furthermore, every satisfying assignment of A must agree with α on all variables $x \notin var(C)$. But there exists no satisfying assignment of $\mathsf{RN}_F(C) \cup \{C\}$ that agrees with $\alpha(x)$ on all $x \notin var(C)$. Hence, $\mathsf{RN}_F(C) \cup A \cup \{C\} = G \wedge C$ is unsatisfiable and thus G and $G \wedge C$ are not equisatisfiable. It follows that C is not redundant with respect to G. $\square$

Theorem 5. $\mathsf{SEM}_{\mathsf{BC}}$ *is the most general local redundancy property.*

Proof. Towards a contradiction, suppose there exists a local redundancy property P that is strictly more general than $\mathsf{SEM}_{\mathsf{BC}}$. It follows that P contains a pair (F, C) such that C is not semantically blocked in F. By Lemma 4, there exists a formula G with $\mathsf{RN}_G(C) = \mathsf{RN}_F(C)$ such that C is not redundant with respect to G. But since P is local and $\mathsf{RN}_G(C) = \mathsf{RN}_F(C)$, it follows that $(G, C) \in \mathsf{P}$, hence P is not a redundancy property, a contradiction. $\square$

2.1.3 Set-Blocked Clauses and Super-Blocked Clauses

In the following, we introduce syntax-based notions of blocking which generalize the original notion of blocking as given in Definition 4. We first present the notion of *set-blocking*, which is already a strict generalization of blocking. After this, we further generalize set-blocking to the so-called notion of *super-blocking* which, as we will prove, coincides with the notion of semantic blocking given in Definition 6. To define set-blocking, we first introduce so-called *set-resolvents*:

Definition 8. *Given two clauses C, D and a set L of literals, the clause $C \otimes_L D = (C \setminus L) \cup (D \setminus \bar{L})$ is the* set-resolvent *of C and D upon L.*

Note that we do not require C and D to contain any of the literals in L or $\bar{L}$. Note also that, in contrast to ordinary resolvents, a set-resolvent is not necessarily implied by its premises.

Definition 9. *A clause C is* set-blocked *in a formula F if it contains a non-empty set L of literals such that for each clause $D \in F_{\bar{L}} \setminus F_L$, the set-resolvent $C \otimes_L D$ is a tautology.*

We say that L set-blocks C in F, and we denote the set $\{(F, C) \mid C$ is set-blocked in $F\}$ by $\mathsf{SET_{BC}}$. It will become clear later why we do not consider set-resolvents with *all* clauses in $F_{\bar{L}}$ but only with the clauses in $F_{\bar{L}} \setminus F_L$. We start with a simple example:

Example 6. *Consider the formula $F = (\bar{x} \vee y) \wedge (\bar{y} \vee x)$ and the clause $(x \vee y)$. Then, $L = \{x, y\}$ trivially set-blocks $(x \vee y)$ in F since $F_{\bar{L}} \setminus F_L$ is empty. Note that $(x \vee y)$ is not blocked in F.*

The following example is a little more involved:

Example 7. *Consider the formula $F = (\bar{x} \vee y) \wedge (\bar{y} \vee x) \wedge (\bar{y} \vee \bar{z}) \wedge (y \vee \bar{z})$ and the clause $(x \vee y \vee z)$. Then, $L = \{x, y\}$ set-blocks $(x \vee y \vee z)$ in F since $F_{\bar{L}} \setminus F_L$ contains only the clause $(\bar{y} \vee \bar{z})$, and the set-resolvent $(x \vee y \vee z) \otimes_L (\bar{y} \vee \bar{z}) = (z \vee \bar{z})$ is a tautology. Note again that $(x \vee y \vee z)$ is not blocked in F.*

As with blocked clauses, we only need to consider the resolution neighborhood of a clause to check if it is set-blocked. Moreover, given an assignment α that satisfies F but falsifies C, the existence of a blocking set L guarantees that the assignment α_L, obtained from α by assigning 1 to all the literals in L, satisfies $F \wedge C$: By assigning 1 to the literals in L, we could only possibly falsify clauses in $F_{\bar{L}}$. However, if a clause contains a literal of L, it will be satisfied by α_L—this is the reason why we do not need to consider set-resolvents with clauses in F_L. Now, let $D \in F_{\bar{L}} \setminus F_L$. Since the set-resolvent of C and D upon L is a tautology, it follows that D contains a literal k such that $\bar{k} \in C \setminus L$. But, since α falsifies C, the assignment α_L must falsify all literals in $C \setminus L$ and thus it must satisfy k. Hence, α_L satisfies D. If follows that α_L satisfies $F \wedge C$ and thus we get:

Theorem 6. $\mathsf{SET}_{\mathsf{BC}}$ *is a local redundancy property.*

Set-blocking is also a strict generalization of blocking:

Theorem 7. *Set-blocking is strictly more general than blocking, i.e.,* $\mathsf{BC} \subset \mathsf{SET}_{\mathsf{BC}}$*.*

Proof. Example 6 shows that $\mathsf{BC} \neq \mathsf{SET}_{\mathsf{BC}}$. It can be easily seen that $\mathsf{BC} \subseteq \mathsf{SET}_{\mathsf{BC}}$: If a clause C is blocked in a formula F, then it contains a literal l such that for each clause $D \in F_{\bar{l}}$, the resolvent $C \otimes_l D$ is a tautology. Thus, for each clause $D \in F_{\bar{l}} \setminus F_l$, the set-resolvent $C \otimes_{\{l\}} D$ is a tautology and so C is set-blocked by $\{l\}$ in F. □

We next generalize set-blocked clauses to obtain *super-blocked clauses*. In their definition, we refer to the *external variables* of a clause, which we define as the variables that occur in the resolution neighborhood of the clause but not in the clause itself:

Definition 10. *Given a formula F and a clause C, the set of* external variables *of C with respect to F, denoted by $ext_F(C)$, is the set $var(\mathsf{RN}_F(C)) \setminus var(C)$.*

We can now define super-blocked clauses:

Definition 11. *A clause C is* super-blocked *in a formula F if, for every assignment α over the external variables $ext_F(C)$, C is set-blocked in $F\,|\,\alpha$.*

We write $\mathsf{SUP}_{\mathsf{BC}}$ for the set $\{(F, C) \mid C$ is super-blocked in $F\}$. In the definition of super-blocked clauses, by "*every assignment α over the external variables $ext_F(C)$*" we mean all assignments whose domain is *exactly* the set $ext_F(C)$.

Example 8. *Consider again the clause $C = (x \vee y)$ from Example 5 and let again F be a formula in which C has the resolution neighborhood $\mathsf{RN}_F(C) = \{(e \vee y \vee \bar{x}), (\bar{y} \vee \bar{e}), (\bar{y} \vee x)\}$. Then, $ext_F(C) = \{e\}$ and thus there are two assignments over the external variables: the assignment e and the assignment $\bar{e}$. The resolution neighborhood of C in $F\,|\,e$ is the set $\{(\bar{y}), (\bar{y} \vee x)\}$. Thus, C is set-blocked by $\{x\}$ in $F\,|\,e$. The resolution neighborhood of C in $F\,|\,\bar{e}$ is the set $\{(y \vee \bar{x}), (\bar{y} \vee x)\}$. Hence, C is set-blocked by $\{x, y\}$ in $F\,|\,\bar{e}$ (see Example 6). It follows that C is super-blocked in F. Note also that C is not set-blocked in F.*

As with the previous redundancy properties, the idea behind super-blocked clauses is again that from an assignment α that satisfies F but falsifies C, we can obtain a satisfying assignment α' of $F \wedge C$ by flipping the truth values of certain literals of C. However, for making sure that the flipping does not falsify any clause D in the resolution neighborhood of C, we are now also allowed to take into account the external literals in D, which could possibly be satisfied by α'. This is in contrast to set-blocking, where we only consider the truth values of literals whose variables occur in C. We show next that super-blocking coincides with semantic blocking:

Theorem 8. *A clause is super-blocked in a formula F if and only if it is semantically blocked in F, i.e., it holds that* $\mathsf{SUP}_{\mathsf{BC}} = \mathsf{SEM}_{\mathsf{BC}}$.

Proof. For the "only if" direction, let C be a clause that is super-blocked in F and let α be a satisfying assignment of $\mathsf{RN}_F(C)$. We have to show that there exists a set $L \subseteq C$ of literals such that α_L satisfies $\mathsf{RN}_F(C) \cup \{C\}$. If α satisfies C, we are done. Assume thus that α does not satisfy C. Furthermore, let $\alpha^{\times}$ be obtained from α by restricting it to the external variables $ext_F(C)$. Since C is super-blocked in F, there exists a non-empty set $L \subseteq C$ that set-blocks C in $F\,|\,\alpha^{\times}$. Now consider the assignment α_L, obtained from α by making all the literals in L true. Clearly, α_L satisfies C and it agrees with $\alpha^{\times}$ on $var(\alpha^{\times}) = ext_F(C)$.

It remains to show that α_L satisfies $\mathsf{RN}_F(C)$. Since α_L and α differ only on L, and since α_L satisfies L, we know that α_L can only possibly falsify clauses in $F_{\bar{L}} \setminus F_L$. Let $D \in F_{\bar{L}} \setminus F_L$. We show that α_L satisfies D. If D is satisfied by $\alpha^{\times}$, then α_L satisfies D since α_L agrees with $\alpha^{\times}$ on $var(\alpha^{\times})$. Assume thus that D is not satisfied by $\alpha^{\times}$. Then, D is contained in $F\,|\,\alpha^{\times}$ and thus, since L set-blocks C in $F\,|\,\alpha^{\times}$, the set-resolvent $C \otimes_L D\,|\,\alpha^{\times}$ is a tautology. Thus, there exists a literal $l \in C \setminus L$ such that $\bar{l} \in D$. But then, since α falsifies C, α_L must falsify l and so α_L must satisfy D. We conclude that C is semantically blocked in F.

For the "if" direction, let F be a formula and let C be a clause that is not super-blocked in F, i.e., there exists an assignment $\alpha^{\times}$ over the external variables $ext_F(C)$ such that C is not set-blocked in $F\,|\,\alpha^{\times}$. We show that C is not semantically blocked in F. First, define the assignment α over the variables in $\mathsf{RN}_F(C) \cup \{C\}$ as follows:

$$\alpha(x) = \begin{cases} 1 & \text{if } \bar{x} \in C, \\ 0 & \text{if } x \in C, \\ \alpha^{\times}(x) & \text{otherwise.} \end{cases}$$

Clearly, α falsifies C and, since (by definition) every clause $D \in \mathsf{RN}_F(C)$ contains a literal $\bar{l}$ such that $l \in C$, it satisfies $\mathsf{RN}_F(C)$. We show that there exists no set $L \subseteq C$ of literals such that α_L satisfies $\mathsf{RN}_F(C) \cup \{C\}$.

Let $L \subseteq C$ be a set of literals. Clearly, α_L agrees with $\alpha^{\times}$ over the external variables $ext_F(C)$, and since C is not set-blocked in $F\,|\,\alpha^{\times}$, there exists a clause $D \in F\,|\,\alpha^{\times}$ such that $D \cap \bar{L} \neq \emptyset$, $D \cap L = \emptyset$, and the set-resolvent $C \otimes_L D$ is not a tautology. Since $D \in F\,|\,\alpha^{\times}$, and since α_L agrees with $\alpha^{\times}$, we know that α_L does not satisfy any of the external variables in D. Thus, D can only possibly be satisfied by α_L if it contains a literal l such that $var(l) \in var(C)$. But α_L satisfies L and falsifies $C \setminus L$, and since $D \cap L = \emptyset$, it follows that α_L could only satisfy D if D contained a literal l such that $\bar{l} \in C \setminus L$, i.e., if the set-resolvent $C \otimes_L D$ were a tautology, which it is not. Hence, α_L falsifies D and so we can conclude that C is not semantically blocked in F. ☐

In Example 8, we have seen a clause that is super-blocked but not set-blocked. Since set-blocking is a local redundancy property and since semantic blocking is the most general local redundancy property, we conclude:

Corollary 9. *Super-blocking is strictly more general than set-blocking, i.e., it holds that* $\mathsf{SET}_{\mathsf{BC}} \subset \mathsf{SUP}_{\mathsf{BC}}$.

2.1.4 Relationship Between Set-Blocked Clauses and Autarkies

Set-blocked clauses are closely related to so-called *autarkies*—a well-known concept from the literature [MS85, KK09]. Intuitively, an autarky is an assignment that satisfies every clause it *touches* (i.e., every clause in which it assigns a truth value to at least one of the literals):

Definition 12. *An assignment* α *is an* autarky *for a formula* F *if* α *satisfies every clause* $C \in F$ *for which* $var(\alpha) \cap var(C) \neq \emptyset$.

Example 9. *Let* $F = (x \vee y \vee \bar{z}) \wedge (\bar{y} \vee z \vee u) \wedge (\bar{x} \vee \bar{u})$ *and let* $\alpha = yz$. *Then,* α *touches only the first two clauses. Since it also satisfies them, it is an autarky for* F.

One of the crucial properties of autarkies, which follows easily from their definition, is that they can be applied to a formula without affecting the formula's satisfiability [KK09]:

Theorem 10. *If* α *is an autarky for a formula* F, *then* F *and* $F|\alpha$ *are equisatisfiable.*

Now, suppose a SAT solver is trying to solve a formula F and that during the solving it obtains a partial assignment α_{c}. If it then detects an autarky α_{a} in the simplified formula $F|\alpha_{\mathrm{c}}$, we can say that α_{a} is an autarky for F under the condition that α_{c} is true. Based on this observation, we generalize the autarky concept as follows:

Definition 13. *An assignment* $\alpha_{\mathrm{c}} \cup \alpha_{\mathrm{a}}$ *(with* $\alpha_{\mathrm{c}} \cap \alpha_{\mathrm{a}} = \emptyset$*) is a* conditional autarky *for a formula* F *if* α_{a} *is an autarky for* $F|\alpha_{\mathrm{c}}$.

We call α_c the *conditional part* and α_{a} the *autarky part* of $\alpha_{\mathrm{c}} \cup \alpha_{\mathrm{a}}$. Observe that every assignment is a conditional autarky with an empty autarky part. We are mainly interested in conditional autarkies with *non-empty* autarky parts:

Example 10. *Consider the formula* $F = (\bar{x} \vee y) \wedge (\bar{y} \vee z) \wedge (x \vee \bar{z}) \wedge (\bar{x} \vee u)$ *and the assignments* $\alpha_{\mathrm{c}} = x$ *and* $\alpha_{\mathrm{a}} = yz$. *The assignment* $\alpha_{\mathrm{c}} \cup \alpha_{\mathrm{a}}$ *is a conditional autarky for* F *since* α_{a} *is an autarky for* $F|\alpha_{\mathrm{c}} = (y) \wedge (\bar{y} \vee z) \wedge (u)$. *Notice that* $\alpha_{\mathrm{c}} \cup \alpha_{\mathrm{a}}$ *is not an autarky for* F *and that also* α_{a} *alone is not an autarky for* F.

Theorem 10 above tells us that the application of an autarky to a formula does not affect the formula's satisfiability. The following statement, which is a simple consequence of Theorem 10 and the fact that α_{a} is an autarky for $F|\alpha_{\mathrm{c}}$, generalizes this statement for conditional autarkies:

Theorem 11. *If $\alpha_c \cup \alpha_a$ is a conditional autarky (with conditional part α_c and autarky part α_a) for a formula F, then $F\,|\,\alpha_c$ and $F\,|\,\alpha_c \cup \alpha_a$ are equisatisfiable.*

Consider now again the formula F and the conditional autarky $\alpha_c \cup \alpha_a$ with $\alpha_c = x$ and $\alpha_a = yz$ from Example 10. Then, the clause $(\bar{x} \vee y \vee z)$, which we obtain by taking the negated literals of α_c together with the literals of α_a, is a set-blocked clause in F: Let $L = \{y, z\}$. Then, $F_{\bar{L}} \setminus F_L = (x \vee \bar{z})$ and the set-resolvent $(\bar{x} \vee y \vee z) \otimes_L (x \vee \bar{z}) = (\bar{x} \vee x)$ is a tautology. This is a consequence of the following theorem, which tells us that every conditional autarky with a non-empty autarky part corresponds to a set-blocked clause:

Theorem 12. *Let F be a formula and let $C = (c_1 \vee \dots \vee c_m \vee l_1 \vee \dots \vee l_n)$ be a clause where $m \geq 0$ and $n \geq 1$. Then, C is set-blocked by $\{l_1, \dots, l_n\}$ in F if and only if the assignment $\bar{c}_1 \dots \bar{c}_m l_1 \dots l_n$ is a conditional autarky for F with conditional part $\bar{c}_1 \dots \bar{c}_m$.*

Proof. For the "only if" direction, assume C is set-blocked by $L = \{l_1, \dots, l_n\}$ in F. We show that α_a is an autarky for $F\,|\,\alpha_c$ with $\alpha_a = l_1, \dots l_n$ and $\alpha_c = \bar{c}_1 \dots \bar{c}_m$. Let $D\,|\,\alpha_c \in F\,|\,\alpha_c$. Then, D is not satisfied by α_c and since α_c falsifies exactly the literals of $C \setminus L$, it follows that D cannot contain the complement of a literal in $C \setminus L$. Hence, the set-resolvent $C \otimes_L D$ is not a tautology and so D cannot be contained in $F_{\bar{L}} \setminus F_L$. Since α_a satisfies exactly the literals in L, this means that if D is touched by α_a, it is also satisfied by α_a. But then, since $\alpha_a \cap \alpha_c = \emptyset$, it cannot be the case that α_a touches $D\,|\,\alpha_c$ without satisfying it. We conclude that α_a is an autarky for $F\,|\,\alpha_c$.

For the "if" direction, assume $\alpha_c \cup \alpha_a$ is a conditional autarky for F with conditional part $\alpha_c = \bar{c}_1 \dots \bar{c}_m$ and autarky part $\alpha_a = l_1 \dots l_n$, and let $L = \{l_1, \dots, l_n\}$. We show that for every clause $D \in F_{\bar{L}} \setminus F_L$, the set-resolvent $C \otimes_L D$ is a tautology. Let $D \in F_{\bar{L}} \setminus F_L$. Then, since α_a falsifies exactly the literals of L, the clause D is touched but not satisfied by α_a. Hence, α_c must satisfy a literal l of D, for otherwise $\alpha_c \cup \alpha_a$ would not be a conditional autarky. Moreover, since α_c assigns no literals of $\bar{L}$, it must actually be the case that $l \in D \setminus \bar{L}$. But then, since α_c falsifies only literals of C, it follows that $\bar{l} \in C$ and so $C \otimes_L D$ is a tautology. We conclude that C is set-blocked by L in F. □

Searching for set-blocked clauses is therefore nothing else than searching for conditional autarkies.

2.1.5 Complexity of Deciding Set-Blockedness and Super-Blockedness

We next analyze the complexity of testing if a clause is set-blocked or super-blocked. We also consider restricted variants of set-blocking and super-blocking, which gives rise to a whole family of blocking notions. Note that all complexity results are with respect to the size of a clause and its resolution neighborhood. This means that even if deciding set-blockedness and super-blockedness is hard in general, it can be easy as long as a clause has a small resolution neighborhood within a formula—no matter how large the formula.

Definition 14. *The* set-blocking problem *is the following decision problem: Given a pair (F, C) where F is a formula and C is a clause such that every clause $D \in F$ contains a literal $\bar{l}$ with $l \in C$, decide if C is set-blocked in F.*

Theorem 13. *The set-blocking problem is NP-complete.*

Proof. We first show NP-membership, followed by NP-hardness.

NP-membership: Given a non-empty set $L \subseteq C$, it can be checked in polynomial time if for every clause $D \in F_{\bar{L}} \setminus F_L$, the set-resolvent $C \otimes_L D$ is a tautology. The following is thus an NP-procedure for the set-blocking problem: Guess a non-empty set $L \subseteq C$ and check if it blocks C in F.

NP-hardness: We give a reduction from the satisfiability problem of propositional logic by defining the following reduction function on the input formula F with $var(F) = \{x_1, \dots, x_n\}$ (we assume without loss of generality that F is in CNF):

$$f(F) = (G, C), \text{ with } C = (u \vee x_1 \vee x_1' \vee \cdots \vee x_n \vee x_n'),$$

where and $u, x_1', \dots, x_n'$ are new variables that do not occur in F. Furthermore, the formula G is obtained from F by

- replacing every clause $C' \in F$ by a clause $t(C')$, obtained from C' by adding the literal $\bar{u}$ and replacing every negative literal $\bar{x}_i$ by the positive literal x_i', and
- adding the clauses $(\bar{x}_i \vee \bar{x}_i'), (\bar{x}_i \vee u), (\bar{x}_i' \vee u)$ for every $x_i \in var(F)$.

The intuition behind the construction of G and C is as follows. By including u in C and adding $\bar{u}$ to every $t(C')$ with $C' \in F$, we guarantee that all clauses in G are in the resolution neighborhood of C, i.e., they contain a literal l with $\bar{l} \in C$. This makes (G, C) a valid instance of the set-blocking problem. The main idea, however, is that every set L that set-blocks C in G corresponds to a satisfying assignment of the original formula F. We show that F is satisfiable if and only if C is set-blocked in G.

For the "only if" direction, suppose there exists a satisfying assignment α of F and let $L = \{u\} \cup \{x_i \mid \alpha(x_i) = 1\} \cup \{x_i' \mid \alpha(x_i) = 0\}$. Clearly, $L \subseteq C$. It remains to show that for every clause $D \in G_{\bar{L}} \setminus G_L$, the set-resolvent $C \otimes_L D$ is a tautology. We proceed by a case distinction.

Case 1: D is of the form $(\bar{x}_i \vee u)$ or $(\bar{x}_i' \vee u)$. Then, $D \notin G_{\bar{L}} \setminus G_L$ since $u \in L$.

Case 2: D is of the form $(\bar{x}_i \vee \bar{x}_i')$. By the definition of L, only one of x_i and x_i' is in L. Assume without loss of generality that $x_i \in L$. Then, $x_i' \in C \setminus L$ and since $\bar{x}_i' \in D$, the set-resolvent $C \otimes_L D$ is a tautology.

Case 3: D is of the form $t(C')$ for $C' \in F$. Since α satisfies F, it must satisfy a literal $l \in C'$. If l is positive, i.e., $l = x_i$ for $x_i \in var(F)$, then (by the construction of $t(C') = D$)

x_i is contained in D and (by the definition of L) x_i is also contained in L. If l is negative, i.e., $l = \bar{x}_i$ for $x_i \in \mathit{var}(F)$, then x'_i is contained in both D and L. In both cases, L contains a literal of D and thus $D \notin G_{\bar{L}} \setminus G_L$.

We conclude that L set-blocks C in G.

For the "if" direction, suppose C is set-blocked by some set L in G, and define α over $\mathit{var}(F)$ as follows:

$$\alpha(x_i) = \begin{cases} 1 & \text{if } x_i \in L, \\ 0 & \text{otherwise.} \end{cases}$$

We show that α satisfies F. First, observe that u must be contained in L: Assume to the contrary that $u \notin L$. Then, since L must be non-empty, some x_i or x'_i must be contained in L. If $x_i \in L$, then $G_{\bar{L}} \setminus G_L$ contains the clause $D = (\bar{x}_i \vee u)$ and so the set-resolvent $C \otimes_L D$ must be a tautology. But this cannot be the case since $\bar{u} \notin C$. The argument for $x'_i \in L$ is analogous.

Now, let $C' \in F$ and let $D = t(C')$. Then, since $u \in L$ and $\bar{u} \in D$, it must be the case that either $D \in G_L$ or the set-resolvent $C \otimes_L D$ is a tautology. But, C contains only positive literals, which is (apart from $\bar{u}$) also the case for D. Hence, $C \otimes_L D$ cannot be a tautology and so D must contain a literal $l \in L$. Now, if $l = x_i$ for $x_i \in \mathit{var}(F)$, then $x_i \in C'$ and $\alpha(x_i) = 1$. If $l = x'_i$ for $x_i \in \mathit{var}(F)$, then $\bar{x}_i \in C'$ and $\alpha(x_i) = 0$. In both cases, α satisfies C'. It follows that α satisfies F. $\square$

We next analyze the complexity of deciding if a clause is super-blocked. To do so, we define the following problem:

Definition 15. *The* super-blocking problem *is the following decision problem: Given a pair (F, C) where F is a formula and C is a clause such that every clause $D \in F$ contains a literal $\bar{l}$ with $l \in C$, decide if C is super-blocked in F.*

Below, we show that the super-blocking problem is Π_2^P-hard by providing a reduction from the $\forall\exists$-SAT problem. The $\forall\exists$-SAT problem takes as input a quantified Boolean formula of the form $\forall X \exists Y.F$ where X and Y are sets of variables and F is a propositional formula. It then asks: Is it the case that for every assignment α^X over the variables in X, there exists an assignment α^Y over the variables in Y such that $\alpha^X \cup \alpha^Y$ satisfies F? We will take a closer look at quantified Boolean formulas later on in Chapter 6.

Theorem 14. *The super-blocking problem is Π_2^P-complete.*

Proof. We show Π_2^P-membership followed by Π_2^P-hardness.

Π_2^P-membership: The following is a Σ_2^P-procedure for deciding if C is *not* super-blocked in F: Guess an assignment α over the external variables $\mathit{ext}_F(C)$ and ask an NP-oracle if C is set-blocked in $F|\alpha$. If the oracle answers *no*, then return *yes*, otherwise return *no*.

Π_2^P*-hardness*: We give a reduction from $\forall\exists$-SAT to the super-blocking problem. The reduction is similar to the one used for proving Theorem 13. The difference is that here we encode only the existentially quantified variables of the $\forall\exists$-formula into C, which makes all the universally quantified variables external. Let $\phi = \forall X \exists Y.F$ be an instance of $\forall\exists$-SAT with $X = \{x_1, \ldots, x_m\}$ and $Y = \{y_1, \ldots, y_n\}$, and assume without loss of generality that F is in CNF. We define the reduction function

$$f(\phi) = (G, C), \text{ with } C = (u \vee y_1 \vee y_1' \vee \cdots \vee y_n \vee y_n'),$$

where $u, y_1', \ldots, y_n'$ are new variables not occurring in ϕ. Furthermore, G is obtained from F by

- replacing every clause $C' \in F$ by a clause $t(C')$ which is obtained from C' by adding the literal $\bar{u}$ and replacing every negative literal $\bar{y}_i$ (where $y_i \in Y$) by the positive literal y_i'; and by
- adding the clauses $(\bar{y}_i \vee \bar{y}_i'), (\bar{y}_i \vee u), (\bar{y}_i' \vee u)$ for every $y_i \in Y$.

By construction, every clause $D \in G$ contains a literal $\bar{l}$ with $l \in C$, hence (G, C) is a valid instance of the super-blocking problem. The intuition behind the reduction is that blocking sets correspond to assignments over the existential variables of ϕ while the assignments over the external variables, $ext_G(C)$, correspond to the assignments over the universally quantified variables of ϕ. We show that ϕ is true if and only if C is super-blocked in G.

For the "only-if" direction, assume that ϕ is true and let α^X be an assignment over the external variables $ext_G(C)$ of C in G. We show that C is set-blocked in $G|\alpha^X$. Since ϕ is true and since $ext_G(C) = X$, there exists an assignment α^Y to the variables in Y such that $\alpha^X \cup \alpha^Y$ satisfies F. Now, let $L = \{u\} \cup \{y_i \mid \alpha^Y(y_i) = 1\} \cup \{y_i' \mid \alpha^Y(y_i) = 0\}$. We show that for every clause $D \in G|\alpha^X$ such that $D \cap \bar{L} \neq \emptyset$ and $D \cap L = \emptyset$, the set-resolvent $C \otimes_L D$ is a tautology. We proceed by a case distinction over the types of clauses in $G|\alpha^X$:

Case 1: D is of the form $(\bar{y}_i \vee u)$ or $\bar{y}_i' \vee u$. Then, D contains u and so $D \cap L \neq \emptyset$.

Case 2: D is of the form $(\bar{y}_i \vee \bar{y}_i')$. By definition, L contains at most one of y_i, y_i'. Assume without loss of generality that $y_i \in L$. Then, $y_i' \in C \setminus L$ and thus the set-resolvent $C \otimes_L D$ is a tautology containing both y_i' and $\bar{y}_i'$. The case when $y_i' \in L$ is analogous.

Case 3: D is of the form $t(C')$ for $C' \in F$. Since D is not satisfied by α^X, it must be satisfied by α^Y and thus α^Y satisfies a literal of C' that is either of the form y_i or of the form $\bar{y}_i$. In the former case, $y_i \in L$ and in the latter case $y_i' \in L$. Hence, in both cases, $D \cap L \neq \emptyset$.

We conclude that C is super-blocked in G.

For the "if" direction, suppose C is super-blocked in G and let α^X be an assignment over the variables in $X = ext_G(C)$. We show that there exists an assignment α^Y over Y such that $\alpha^X \cup \alpha^Y$ satisfies F. Since C is super-blocked in G, there exists a non-empty set L that set-blocks C in $G|\alpha^X$. We define the assignment α^Y as follows:

$$\alpha^Y = \begin{cases} 1 & \text{if } y_i \in L, \\ 0 & \text{otherwise.} \end{cases}$$

It remains to show that $\alpha^X \cup \alpha^Y$ satisfies F. First, observe that u must be contained in L: Assume to the contrary that $u \notin L$. Then, since L must be non-empty, some y_i or y_i' must be contained in L. If $y_i \in L$, then the set-resolvent $C \otimes_L D$ with the clause $D = (\bar{y}_i \vee u)$, which is contained in $G|\alpha^X$, must be a tautology. But this cannot be the case since $\bar{u} \notin C$. The argument for $y_i' \in L$ is analogous.

Second, observe that L cannot contain both y_i and y_i' since in that case the set-resolvent of C with the clause $(\bar{y}_i \vee \bar{y}_i')$ upon L cannot contain two complementary literals (both $\bar{y}_i$ and $\bar{y}_i'$ are not contained in the set-resolvent). Hence, if $y_i' \in L$, then $\alpha^Y(y_i) = 0$.

Now, let $C' \in F$ be a clause that is not satisfied by α^X. Then, G contains the clause $D = t(C')$. Since $u \in L$ and since every clause $t(C') \in G$ contains the literal $\bar{u}$, it must be the case that either D contains a literal $l \in L$ or the set-resolvent $C \otimes_L D|\alpha^X$ is a tautology. But this set-resolvent cannot be a tautology since both C and $D|\alpha$ contain (apart from u) only positive literals of the form y_i or y_i'. It follows that D contains a literal $l \in L$. Now, l is either of the form y_i or of the form y_i'. In the former case $y_i \in C'$ is satisfied by α^Y. In the latter case, $\bar{y}_i \in C'$ is satisfied by α^Y. Thus, α^Y satisfies all clauses of F that are not satisfied by α^X. We conclude that ϕ is true. $\square$

We have already seen that the set-blocking problem is NP-complete in the general case. However, we obtain a restricted variant of set-blocking by only allowing blocking sets whose size is bounded by a constant. Then, the resulting problem of testing if a clause C is blocked by some non-empty set $L \subseteq C$ whose size is at most k (for $k \in \mathbb{N}^+$) turns out to be polynomial: For a finite set C and $k \in \mathbb{N}^+$, there are only polynomially many non-empty subsets $L \subseteq C$ with $|L| \leq k$. To see this, let $n = |C|$ and observe (by basic combinatorics) that the exact number of such subsets is given by the following sum which reduces to a polynomial with degree at most k:

$$\sum_{i=1}^{k} \binom{n}{i}.$$

Hence, the number of non-empty subsets $L \subseteq C$ with $|L| \leq k$ is polynomial in the size of C. This line of argumentation is actually very common. For the sake of illustration, however, we provide the following example:

Example 11. *Let $|C| = n$ and $k = 3$ (with $k \leq n$). Then, the number of non-empty subsets $L \subseteq C$ with $|L| \leq k$ is given by the polynomial*

$$\sum_{i=1}^{3} \binom{n}{i} = \frac{n(n-1)(n-2)}{6} + \frac{n(n-1)}{2} + n = \frac{1}{6}n^3 + \frac{5}{6}n.$$

As there are only polynomially many potential blocking sets and since we can easily check in polynomial time if a given set L set-blocks C in F, we can check in polynomial time if for some clause C there exists a blocking set L of size at most k.

Since the definition of super-blocking is based on the definition of set-blocking, we can also consider the complexity of restricted variants of super-blocking where again the size of the according blocking sets is bounded by a constant. We thus define an infinite number of decision problems (one for every $k \in \mathbb{N}^+$) as follows:

Definition 16. *For any $k \in \mathbb{N}^+$, the* k-super-blocking problem *is the following decision problem: Given a pair (F, C) where F is a formula and C is a clause such that every clause $D \in F$ contains a literal $\bar{l}$ with $l \in C$, decide if for every assignment α over the external variables $ext_F(C)$, there exists a non-empty set $L \subseteq C$ with $|L| \leq k$ that set-blocks C in $F|\alpha$.*

Theorem 15. *The k-super-blocking problem is in co-NP for all $k \in \mathbb{N}^+$.*

Proof. Consider the statement that has to be tested for the complement of the k-super-blocking problem:

> There exists an assignment α over the external variables $ext_F(C)$ such that *no* non-empty subset L of C with $|L| \leq k$ set-blocks C in $F|\alpha$.

Since we can easily check in polynomial time if a given set $L \subseteq C$ set-blocks C in $F|\alpha$, the following is an NP-procedure for this complementary problem:

> Guess an assignment α over the external variables $ext_F(C)$ and check for every non-empty subset L of C (with $|L| \leq k$) if it set-blocks C in $F|\alpha$. If there is such a set, return *no*, otherwise return *yes*.

Hence, for every $k \in \mathbb{N}^+$, the k-super-blocking problem is in co-NP. $\square$

We can show co-NP-hardness of the k-super-blocking problem already for $k = 1$:

Theorem 16. *The 1-super-blocking problem is co-NP-hard.*

Proof. We show the result by providing a reduction from the unsatisfiability problem of propositional logic. Let $F = C_1 \wedge \cdots \wedge C_n$ be a propositional formula in CNF and define the reduction function

$$f(F) = (G, C), \text{ with } C = (u_1 \vee \cdots \vee u_n),$$

where $u_1, \dots, u_n$ are new variables that do not occur in F, and

$$G = \bigwedge_{i=1}^{n} F_i \quad \text{where} \quad F_i = \bigwedge_{l \in C_i} (\bar{u}_i \vee \bar{l}).$$

Clearly, (G, C) is a valid instance of the 1-super-blocking problem and $var(F) = ext_G(C)$. We show that F is unsatisfiable if and only if, for every assignment α over the external variables $ext_G(C)$, there exists a literal $u_i \in C$ such that $\{u_i\}$ set-blocks C in $G|\alpha$.

For the "only if" direction, assume that F is unsatisfiable and let α be an assignment over $ext_G(C)$. Since $var(F) = ext_G(C)$, there exists a clause C_i in F that is falsified by α. But then, since every clause in F_i contains a literal $\bar{l}$ with $l \in C_i$, it follows that F_i is satisfied by α. Hence, as $\bar{u}_i$ only occurs in F_i, $\{u_i\}$ trivially set-blocks C in $G|\alpha$.

For the "if" direction, assume that for every α over $ext_G(C)$, there exists a literal $u_i \in C$ such that $\{u_i\}$ set-blocks C in $G|\alpha$. Now, let α be an assignment over $var(F) = ext_G(C)$. Since no clause in G contains a literal l such that $\bar{l} \in C \setminus \{u_i\}$, none of the set-resolvents of C upon $\{u_i\}$ with clauses in $G|\alpha$ can be tautologies. Hence, α must satisfy every clause $(\bar{u}_i \vee \bar{l}) \in G$ and thus it must falsify every literal $l \in C_i$. It follows that every assignment over $var(F)$ must falsify a clause in F and thus F is unsatisfiable. ☐

The above reduction actually works for all k-super-blocking-problems with $k \in \mathbb{N}^+$. To see this, observe that for every $k \in \mathbb{N}^+$, C is k-super-blocked in G if and only if it is 1-super-blocked in G: If a clause is 1-super-blocked in a formula, then it is by definition also k-super-blocked for all $k \in \mathbb{N}^+$. Conversely, due to the way G is constructed, if a set $L \subseteq C$ set-blocks C in $G|\alpha$ (with α being an arbitrary assignment over $ext_G(C)$), then there exists a singleton set $L' \subseteq L$ that set-blocks C in $G|\alpha$ and thus C is 1-super-blocked in G. We thus get:

Corollary 17. *The k-super-blocking problem is co-NP-complete for all $k \in \mathbb{N}^+$.*

The notions of set-blocking and super-blocking, together with the corresponding restrictions discussed in this section, give rise to a whole family of blocking notions which differ in both generality and complexity. We conclude the following:

1. Taking the assignments over external variables into account (as is the case for super-blocking) leads to co-NP-hardness.
2. If blocking sets of arbitrary size are considered, the (sub-)problem of checking if there exists a blocking set is NP-hard.
3. If the size of blocking sets is bounded by a constant k, the (sub-)problem of testing if there exists a blocking set turns out to be polynomial.
4. The problem of testing if a clause is super-blocked in the most general sense, where the size of blocking sets is not bounded by a constant, is Π_2^P-complete.

Hence, we can summarize the following complexity results:

	$\lvert L\rvert$ is unrestricted	$\lvert L\rvert \leq k$ for $k \in \mathbb{N}^+$
Super-Blocking	Π_2^P-complete	co-NP-complete
Set-Blocking	NP-complete	P

Note that the cardinality $|L|$ of blocking sets is of course bounded by the length of the clauses, thus we can restrict $|L| \leq |C|$. This is particularly interesting for formula instances with (uniform) constant or maximal clause length.

After having seen different notions of locally redundant clauses, we now drop the restriction of locality and consider redundancy properties that can require us to consider a whole formula instead of only the resolution neighborhood of a clause.

2.2 Globally Redundant Clauses

In the following, we first review some global redundancy properties from the literature. After this, we come up with a characterization of clause redundancy that is based on a semantic implication relationship between formulas. By replacing the implication relation in this characterization with restricted notions of implication that are computable in polynomial time, we then obtain powerful global redundancy properties that are still efficiently decidable. These redundancy properties not only generalize existing ones such as *resolution asymmetric tautologies* [JHB12] or *set-blocked clauses* but they are also closely related to other concepts from the literature, including *autarkies* [MS85], *safe assignments* [WFS06], and *variable instantiation* [ABCH02].

2.2.1 Globally Redundant Clauses From the Literature

We start by considering a redundancy property from the literature that plays an important role in practical SAT solving—the redundancy property RUP (short for *reverse unit propagation*) [VG12b]. To do so, we require the notion of *unit propagation*, which will play an important role throughout the following chapters.

Unit propagation is based on the repeated application of the *unit-clause rule*: Given a formula F that contains a unit clause (x), the result of applying the unit-clause rule to F is the formula $F\,|\,x$. We also refer to applications of the unit-clause rule as *unit-propagation steps*. *Unit propagation* is then the iterated application of the unit-clause rule to a formula until no unit clauses are left. If unit propagation on F yields the empty clause $\bot$, we say that it *derives a conflict* on F. Note that this definition of unit propagation is non-deterministic in the sense that we are free to choose which unit clause we pick for a single application of the unit-clause rule. It is, however, well-known that the choice of unit clauses does not affect whether unit propagation derives a conflict. In other words, if unit propagation derives a conflict, it does so independently of the order in which we apply the unit-propagation steps.

Example 12. *Consider the formula* $F = (\bar{x} \vee y) \wedge (\bar{y}) \wedge (x)$. *As* F *contains the two unit clauses* $(\bar{y})$ *and* (x), *we can apply the unit-clause rule with either of them. Say we first choose* (x), *then we obtain* $F\,|\,x = (y) \wedge (\bar{y})$. *Next, we can choose either* (y) *or* $(\bar{y})$. *In any case, we obtain* $F\,|\,xy = F\,|\,x\bar{y} = \bot$ *and thus unit propagation derives a conflict on* F.

Clearly, if unit propagation derives a conflict on a formula, then that formula is unsatisfiable, but the converse does not hold. A simple example of an unsatisfiable formula on which unit propagation does not derive a conflict is the formula $(x \vee y) \wedge (\bar{x} \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{x} \vee \bar{y})$.

The concepts of unsatisfiability and implication are closely related: A formula F implies a clause $(l_1 \vee \cdots \vee l_k)$ if and only if the formula $F \wedge (\bar{l}_1) \wedge \cdots \wedge (\bar{l}_k)$ is unsatisfiable. As the satisfiability problem of propositional logic is NP-complete, the problem of deciding if a clause is implied by a formula is co-NP-complete. However, an efficiently decidable notion of implication can be obtained by requiring that unit propagation (which can be performed in polynomial time) must derive a conflict on $F \wedge (\bar{l}_1) \wedge \cdots \wedge (\bar{l}_k)$. This leads to the redundancy property RUP, which can be seen as a restricted form of implication [VG12b]:

Definition 17. *A clause* $(l_1 \vee \cdots \vee l_k)$ *is a* RUP *in a formula* F *if unit propagation derives a conflict on* $F \wedge (\bar{l}_1) \wedge \cdots \wedge (\bar{l}_k)$.

We overload notation by denoting the set $\{(F, C) \mid C \text{ is a } \mathsf{RUP} \text{ in } F\}$ by RUP. If C is a RUP in F, we write $F \vdash_1 C$ and we say that F *implies* C *via unit propagation*. We also say that a formula F implies a formula G via unit propagation, denoted by $F \vdash_1 G$, if $F \vdash_1 D$ holds for each clause $D \in G$.

Example 13. *The formula* $(\bar{x} \vee z) \wedge (\bar{y} \vee \bar{z})$ *implies the clause* $(\bar{x} \vee \bar{y})$ *via unit propagation since unit propagation derives a conflict on* $(\bar{x} \vee z) \wedge (\bar{y} \vee \bar{z}) \wedge (x) \wedge (y)$.

Observe that if C is a resolvent of two clauses in a formula F, or if C is subsumed in F, then C is a RUP in F. Moreover, if C is a RUP in F, then F implies C. Clearly, RUP clauses are redundant [VG12b]:

Theorem 18. RUP *is a redundancy property.*

The RUP redundancy property is non-local since it can take the whole formula into account when checking if unit propagation derives a conflict. An alternative characterization of RUPs are so-called *asymmetric tautologies*, whose definition is based on *asymmetric literals* [HJB10a]:

Definition 18. *A literal* l *is an* asymmetric literal *with respect to a clause* C *in a formula* F *if there exists a clause* $D \vee \bar{l} \in F$ *such that* $D \subseteq C$.

An asymmetric tautology is then a clause that can be turned into a tautology by repeatedly adding asymmetric literals [HJB10a]:

Definition 19. *A clause C is an* asymmetric tautology *in a formula F if there exists a sequence $l_1, \dots, l_n$ of literals such that $C \lor l_1 \lor \cdots \lor l_n$ is a tautology and each l_i is an asymmetric literal with respect to $C \lor l_1 \lor \cdots \lor l_{i-1}$ in F.*

Example 14. *Consider again the formula $F = (\bar{x} \lor z) \land (\bar{y} \lor \bar{z})$ and the clause $(\bar{x} \lor \bar{y})$ from Example 13. The literal $\bar{z}$ is an asymmetric literal with respect to $(\bar{x} \lor \bar{y})$ in F since $(\bar{x}) \subseteq (\bar{x} \lor \bar{y})$. We thus add it to $(\bar{x} \lor \bar{y})$ to obtain $(\bar{x} \lor \bar{y} \lor \bar{z})$. The literal z is an asymmetric literal with respect to $(\bar{x} \lor \bar{y} \lor \bar{z})$ since $(\bar{y}) \subseteq (\bar{x} \lor \bar{y} \lor \bar{z})$. After adding z to $(\bar{x} \lor \bar{y} \lor \bar{z})$ we obtain the tautology $(\bar{x} \lor \bar{y} \lor \bar{z} \lor z)$ and thus $(\bar{x} \lor \bar{y})$ is an asymmetric tautology in F.*

Although we are not aware of a publication where this is proved explicitly, it is well-known that RUPs and asymmetric tautologies coincide. Formally, this can be shown by an easy induction argument. Intuitively, asymmetric-literal additions correspond to applications of the unit-clause rule, and the presence of two complementary literals in an asymmetric tautology corresponds to the derivation of a conflict during unit propagation:

Theorem 19. *A clause is an asymmetric tautology in a formula F if and only if it is a* RUP *in F.*

It follows that asymmetric tautologies are redundant. We denote the redundancy property $\{(F, C) \mid C$ is an asymmetric tautology in $F\}$ by AT.

Clearly, every tautology is an asymmetric tautology, but by adding asymmetric literals, we can even turn non-tautological clauses into tautologies. Asymmetric tautologies are thus a generalization of ordinary tautologies. Now, remember that a blocked clause is a clause for which all resolvents upon one of its literals are tautologies. By slightly modifying this definition, replacing *tautologies* by *asymmetric tautologies*, we arrive at the notion of a *resolution asymmetric tautology*, better known as RAT [JHB12]:

Definition 20. *A clause C is a* resolution asymmetric tautology (RAT) *in a formula F if it contains a literal l such that for every clause $D \in F_{\bar{l}}$, the resolvent $C \otimes_l D$ is an asymmetric tautology in F.*

We say that C is a RAT *on l* in F. Again, we overload notation, referring to the set $\{(F, C) \mid C$ is a resolution asymmetric tautology in $F\}$ by RAT. Since every tautology is an asymmetric tautology, but not vice versa, the RAT redundancy property is a strict generalization of blocked clauses.

Example 15. *Consider the formula $F = (\bar{x} \lor \bar{y}) \land (\bar{x} \lor z) \land (z \lor u) \land (\bar{u} \lor y)$ and the clause $(x \lor y)$. There are two resolvents of $(x \lor y)$ upon x: The resolvent $(y \lor \bar{y})$, obtained by resolving with $(\bar{x} \lor \bar{y})$, is a tautology in F; the resolvent $(y \lor z)$, obtained by resolving* *with $(\bar{x} \lor z)$ is not a tautology, but it is an asymmetric tautology in F: Using the clause $(z \lor u)$, we can add the asymmetric literal $\bar{u}$ to $(y \lor z)$. After this, we can use the clause $(\bar{u} \lor y)$ to add the asymmetric literal u to obtain the tautology $(y \lor z \lor \bar{u} \lor u)$. It follows that $(x \lor y)$ is a* RAT *on x in F. Note that $(x \lor y)$ is not blocked in F.*

It can be shown that if a clause C is a RAT in a formula F, then C is redundant with respect to F, which means that RAT is a redundancy property [JHB12]. The main idea behind the redundancy of RATs is similar to the idea behind the redundancy of blocked clauses: If a clause C is a RAT on a literal l in a formula F, then every satisfying assignment of F that falsifies C can be turned into a satisfying assignment of $F \land C$ by flipping the truth value of l. The condition that all resolvents of C upon l are asymmetric tautologies guarantees that this does not affect the truth of clauses in F.

The RAT redundancy property not only generalizes blocked clauses but also several other redundancy properties from the literature [JHB12]. Moreover, by adding and removing RATs, it is possible to simulate most of the reasoning techniques employed by state-of-the-art SAT solvers. Because of this, RAT provides the basis for the well-known DRAT proof system [WHHJ14], which is the de facto standard for unsatisfiability proofs in practical SAT solving (for a formal definition of DRAT, see page 48). Participants in the annual SAT competition, where the best SAT solvers compete against each other, are required to produce DRAT proofs [BHJ17]. Also, recent proofs of open mathematical problems, including the Erdős Discrepancy Conjecture [KL15] and the Pythagorean Triples Problem [HKM16], were provided in DRAT.

Since asymmetric tautologies and RUPs coincide, we get the following alternative characterization of RATs, which is sometimes used in the literature as *the* RAT definition:

Theorem 20. *A clause C is a* RAT *in a formula F if and only if it contains a literal l such that for every clause $D \in F_{\bar{l}}$, the resolvent $C \otimes_l D$ is a* RUP *in F.*

We have now seen two different approaches to generalizing blocked clauses: On the one hand, we have set-blocked clauses and super-blocked clauses. They generalize blocked clauses by allowing us to modify the truth values of multiple literals when showing that they are redundant. On the other hand, we have RATs, which—like blocked clauses—only allow us to modify the truth value of a single literal. However, unlike set-blocked clauses and super-blocked clauses, RATs go beyond the resolution neighborhood of clauses when it comes to showing their redundancy. It turns out that the redundancy properties of both set-blocked clauses and super-blocked clauses are incomparable with RAT. This means that there are clauses that are set-blocked (or super-blocked) with respect to certain formulas while they are not RATs, and vice versa.

Theorem 21. $\mathsf{RAT} \not\subseteq \mathsf{SET}_{\mathsf{BC}}$.

Proof. Consider the formula $F = (\bar{x} \lor y) \land (\bar{y} \lor x)$ and the clause $(x \lor y)$ from Example 6. The set $L = \{x, y\}$ trivially set-blocks $(x \lor y)$ in F since $F_{\bar{L}} \setminus F_L$ is empty. However,

$(x \vee y)$ is not a RAT on x in F since the resolvent (y), obtained by resolving $(x \vee y)$ with $(\bar{x} \vee y)$ upon x, is not a RUP in F. Moreover, $(x \vee y)$ is also not a RAT on y since the resolvent (x), obtained by resolving $(x \vee y)$ with $(\bar{y} \vee x)$ upon y, is not a RUP in F. □

Theorem 22. $\mathsf{SUP_{BC}} \not\subseteq \mathsf{RAT}$.

Proof. Consider the formula $F = (\bar{x} \vee e) \wedge (\bar{y} \vee e) \wedge (\bar{z} \vee e) \wedge (x \vee y)$ and the clause $C = (x \vee y \vee z)$. It is easy to see that C is a RAT on z in F: There exists only one resolvent of C upon z, namely the clause $(x \vee y \vee e)$, obtained by resolving with $(\bar{z} \vee e)$. This resolvent is a RUP in F since F contains the clause $(x \vee y)$, and unit propagation derives a conflict on $(x \vee y) \wedge (\bar{x}) \wedge (\bar{y}) \wedge (\bar{e})$.

It remains to show that C is not super-blocked in F. To do so, we show that for the assignment $\bar{e}$ over the external variables $ext_F(C) = \{e\}$, C is not set-blocked in $F\,|\,\bar{e} = (\bar{x}) \wedge (\bar{y}) \wedge (\bar{z}) \wedge (x \vee y)$. Assume to the contrary that C is set-blocked in $F\,|\,\bar{e}$. This means that there exists a non-empty set $L \subseteq C$ of literals such that for every clause $D \in F\,|\,\bar{e}$ with $D \cap \bar{L} \neq \emptyset$ and $D \cap L = \emptyset$, the set-resolvent $C \otimes_L D$ is a tautology. Since L is non-empty, L must contain at least one of the literals x, y, and z. Assume without loss of generality that $x \in L$. Then, the only literals that can be contained in the set-resolvent $C \otimes_L (\bar{x})$ are y and z, implying that $C \otimes_L (\bar{x})$ cannot be a tautology. The cases where $y \in L$ or $z \in L$ are analogous. It follows that C is not set-blocked in $F\,|\,\bar{e}$ and thus C is not super-blocked in F. □

Note that the proof actually shows a stronger result, namely that $\mathsf{SUP_{BC}} \not\subseteq \mathsf{RS}$, where $\mathsf{RS} = \{(F, C) \mid \text{all resolvents of } C \text{ upon one of its literals are subsumed in } F\}$ [JHB12], which is a strict subset of RAT.

Since every set-blocked clause is a super-blocked clause, Theorem 22 allows us to conclude:

Corollary 23. RAT *is incomparable with both set-blocked clauses* ($\mathsf{SET_{BC}}$) *and super-blocked clauses* ($\mathsf{SUP_{BC}}$).

In the following, we introduce redundancy properties that generalize both set-blocked clauses and RATs. As our complexity analysis in Section 2.1.5 has revealed that deciding super-blockedness is extremely hard, we do not introduce any generalizations of super-blocked clauses.

2.2.2 Characterizing Clause Redundancy via Implication

We want to combine the ideas behind both set-blocked clauses and RATs to obtain even stronger redundancy properties. To achieve this, we introduce a characterization of clause redundancy that reduces the question if a clause is redundant to a question of implication between two formulas. The advantage of this is that we can then replace the ordinary implication relation in this characterization by polynomially decidable implication relations to derive powerful redundancy properties that are still efficiently checkable. We use these redundancy properties later to obtain highly expressive clausal proof systems.

Our characterization is based on the observation that a clause can be seen as a constraint that rules out those assignments that falsify the clause. For instance, if a formula contains the clause $(x \lor \bar{y})$, then the formula cannot be satisfied by any assignment that falsifies x and satisfies y. We thus say that $(x \lor \bar{y})$ *precludes* the assignment $\bar{x}y$. More generally:

Definition 21. *Given an assignment $\alpha = a_1 \dots a_k$, the clause $(\bar{a}_1 \lor \dots \lor \bar{a}_k)$ is the clause that* precludes *α.*

Intuitively, a clause is redundant with respect to a formula if its addition does not constrain the formula too much. What we mean by this is that after adding the clause, there should still exist other assignments (i.e., assignments not precluded by the clause) under which the formula is at least as satisfiable as under the assignments precluded by the clause. But when is a formula *at least as satisfiable* as another formula? We say that a formula F is at least as satisfiable as a formula G if every satisfying assignment of F is also a satisfying assignment of G, i.e., if $F \models G$. Consider the following example:

Example 16. *Consider the formula $F = (x \lor y) \land (x \lor z) \land (\bar{x} \lor y \lor z)$ and the unit clause (x). Although the addition of (x) to F precludes the assignment $\alpha = \bar{x}$, there still exists another assignment under which F is at least as satisfiable as under α, namely the assignment $\omega = x$: Observe that $F|_\alpha = (y) \land (z)$ while $F|_\omega = (y \lor z)$, and so every satisfying assignment of $F|_\alpha$ is also a satisfying assignment of $F|_\omega$, that is, $F|_\alpha \models F|_\omega$. Thus, F is at least as satisfiable under ω as it is under α. Moreover, ω satisfies (x). The addition of (x) does therefore not affect the satisfiability of F.*

This motivates our new characterization of clause redundancy presented next. Note that the assignment α precluded by a given clause C is in general a partial assignment and thus C eliminates all assignments that extend α from the search space:

Theorem 24. *Let F be a formula, C a non-empty clause, and α the assignment precluded by C. Then, C is redundant with respect to F if and only if there exists an assignment ω such that ω satisfies C and $F|_\alpha \models F|_\omega$.*

Proof. For the "only if" direction, assume that C is redundant with respect to F, meaning that F and $F \land C$ are equisatisfiable. If $F|_\alpha$ is unsatisfiable, then $F|_\alpha \models F|_\omega$ for every assignment ω, hence the statement trivially holds. Assume now that $F|_\alpha$ is satisfiable, implying that F is satisfiable. Then, since F and $F \land C$ are equisatisfiable, there exists an assignment ω that satisfies both F and C. Hence, since ω satisfies F, it holds that $F|_\omega = \emptyset$ and so $F|_\alpha \models F|_\omega$.

For the "if" direction, suppose there exists an assignment ω such that ω satisfies C and $F|_\alpha \models F|_\omega$. Now, let γ be a (total) assignment that satisfies F and falsifies C. We show how γ can be turned into a satisfying assignment γ' of $F \land C$. As γ falsifies C, it agrees with α on $var(\alpha)$. Therefore, since γ satisfies F, it must satisfy $F|\alpha$ and since $F|\alpha \models F|\omega$ it must also satisfy $F|\omega$. We now define the following assignment which satisfies $F \wedge C$:

$$\gamma'(x) = \begin{cases} \omega(x) & \text{if } x \in var(\omega), \\ \gamma(x) & \text{otherwise.} \end{cases}$$

Clearly, since ω satisfies C, γ' also satisfies C. Moreover, as γ satisfies $F|\omega$, and since $var(F|\omega) \subseteq var(\gamma) \setminus var(\omega)$, γ' satisfies F. We conclude that γ' satisfies $F \wedge C$. $\square$

This alternative characterization of clause redundancy allows us to replace the logical implication relation by restricted implication relations that are polynomially decidable. We can, for instance, replace the condition $F|\alpha \models F|\omega$ by the restricted condition $F|\alpha \vdash_1 F|\omega$ (implication via unit propagation, as defined on page 30). Likewise, we could also replace " $\models$" by relations such as "$\supseteq$", "$=$", or the relation "$\vdash_0$", where $F \vdash_0 G$ denotes that every clause of G is subsumed in F.

As an example, consider blocked clauses: If C is a clause that is blocked by a literal l in a formula F and if α is the assignment precluded by C, then one can show that $F|\alpha \supseteq F|\alpha_l$ (see proof of Theorem 28 on page 39 for details).

Now, if we are given a clause C—which implicitly gives us the precluded assignment α—and a *witnessing assignment* ω, we can check in polynomial time if $F|\alpha \vdash_1 F|\omega$. This gives rise to *propagation-redundant clauses*, which we introduce next.

2.2.3 Propagation-Redundant Clauses

In the following, we use the propagation-implication relation " $\vdash_1$" to define the redundancy properties of

- *literal-propagation redundancy* (LPR),
- *set-propagation redundancy* (SPR),
- *propagation redundancy* (PR).

Basically, the three notions differ in the way we allow the witnessing assignment ω to differ from the assignment α precluded by a clause. The more freedom we give to ω, the more general the redundancy property we obtain. We show that literal-propagation-redundant clauses—the least general of the three—coincide with RAT. For the more general set-propagation-redundant clauses, we show that they not only generalize RAT but also set-blocked clauses ($\mathsf{SET_{BC}}$), which is not the case for literal-propagation-redundant clauses. Finally, propagation-redundant clauses are even more general than set-propagation-redundant clauses. They give rise to an extremely powerful proof system.

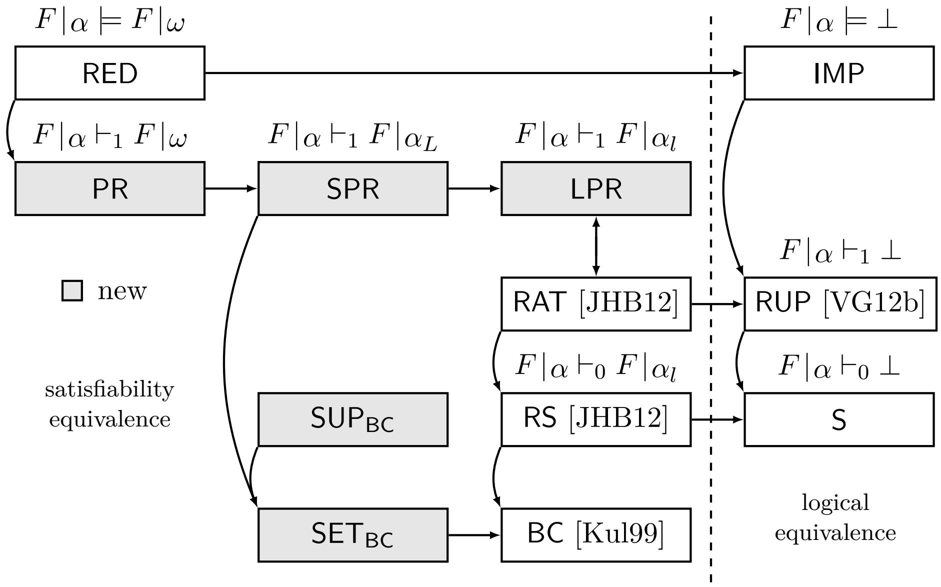


Figure 2.2: Landscape of redundancy properties. A path from a redundancy property X to a redundancy property Y indicates that X is more general than Y.

With these three notions, we obtain the landscape of redundancy properties illustrated in Figure 2.2. In the figure, S stands for the set $\{(F, C) \mid C$ is subsumed in $F\}$, IMP for $\{(F, C) \mid F \models C\}$, and RED for $\{(F, C) \mid C$ is redundant with respect to $F\}$.

As we will see, when defining proof systems based on literal-propagation-redundant clauses (for example, the DRAT proof system) or set-propagation-redundant clauses, we do not need to explicitly add the redundancy witnesses (i.e., the witnessing assignments ω) to a proof. Thus, proofs in the respective proof systems can just be seen as sequences of clauses. In particular, a proof system based on set-propagation-redundant clauses can have the same syntax as DRAT proofs, which makes it “downwards compatible” with DRAT. This is in contrast to proof systems based on propagation-redundant clauses, where in general witnessing assignments have to be added to a proof; otherwise redundancy of a clause cannot be checked in polynomial time. We start by introducing literal-propagation-redundant clauses:

Definition 22. *Let F be a formula, C a clause, and α the assignment precluded by C. Then, C is* literal-propagation redundant (LPR) *with respect to F if there exists a literal $l \in C$ such that $F\,|\,\alpha \vdash_1 F\,|\,\alpha_l$.*

We denote the set $\{(F, C) \mid C$ is literal-propagation redundant with respect to $F\}$ by LPR. It is a straightforward consequence of Theorem 24 that LPR is a redundancy property.

Example 17. *Let* $F = (x \vee y) \wedge (x \vee \bar{y} \vee z) \wedge (\bar{x} \vee z)$ *and let* C *be the unit clause* (x). *Then,* $\alpha = \bar{x}$ *is the assignment precluded by* C*, and* $\alpha_x = x$. *Now, consider* $F\,|\,\alpha = (y) \wedge (\bar{y} \vee z)$ *and* $F\,|\,\alpha_x = (z)$. *Clearly,* $F\,|\,\alpha \vdash_1 F\,|\,\alpha_x$ *and therefore* C *is literal-propagation redundant with respect to* F.

The LPR definition is quite restrictive since it requires the witnessing assignment α_l to disagree with α on exactly one variable. Nevertheless, this already suffices for LPR to coincide with RAT:

Theorem 25. *A clause* C *is literal-propagation redundant with respect to a formula* F *if and only if it is a* RAT *in* F.

Proof. For the "only if" direction, assume that C is literal-propagation redundant with respect to F and let α be the assignment precluded by C. Then, C contains a literal l such that $F\,|\,\alpha \vdash_1 F\,|\,\alpha_l$. Now, let $D \in F_{\bar{l}}$. We show that F implies the resolvent $C \otimes_l D$ via unit propagation, i.e., $F \vdash_1 C \otimes_l D$. Since $F\,|\,\alpha \vdash_1 F\,|\,\alpha_l$, either $D\,|\,\alpha_l = \top$ or $F\,|\,\alpha \vdash_1 D\,|\,\alpha_l$. In case $D\,|\,\alpha_l = \top$, the clause $D \setminus \{\bar{l}\}$ must contain a literal $\bar{d}$ such that $d \in C \setminus \{l\}$ and thus unit propagation on $C \otimes_l D$ alone derives a conflict.

Consider now the case when $F\,|\,\alpha \vdash_1 D\,|\,\alpha_l$. First, note that C is of the form $(c_1 \vee \cdots \vee c_i \vee l)$, α is then of the form $\bar{c}_1 \ldots \bar{c}_i \bar{l}$, and D is of the form $(d_1 \vee \cdots \vee d_j \vee \bar{l})$. We show that unit propagation derives a conflict on $F \wedge (\bar{c}_1) \wedge \cdots \wedge (\bar{c}_i) \wedge (\bar{d}_1) \wedge \cdots \wedge (\bar{d}_j)$. By applying the unit-clause rule with the unit clauses $(\bar{d}_1), \ldots, (\bar{d}_j)$, we derive either a conflict or the unit clause $(\bar{l})$ because $D \in F$. If we do not derive a conflict, we can continue to apply the unit-clause rule, starting with the unit clauses $(\bar{c}_1), \ldots, (\bar{c}_i), (\bar{l})$. This must eventually derive a conflict since $F\,|\,c_1 \ldots c_i \bar{l} = F\,|\,\alpha$ and since $F\,|\,\alpha \vdash_1 D\,|\,\alpha_l$ with $D\,|\,\alpha_l \subseteq D$. It follows that C is a RAT in F.

For the "if" direction, suppose C is a RAT in F, meaning that C contains a literal l such that for every clause $D \in F_{\bar{l}}$, it holds that $F \vdash_1 C \otimes_l D$. Now, let α be the assignment precluded by C and let $D\,|\,\alpha_l \in F\,|\,\alpha_l$ for $D \in F$. We have to show that $F\,|\,\alpha \vdash_1 D\,|\,\alpha_l$. Since $D\,|\,\alpha_l \in F\,|\,\alpha_l$, we know that α_l does not satisfy D. Thus, since α_l satisfies l and since α falsifies C, the clause D does neither contain l nor the negations of any other literals in C, except for possibly $\bar{l}$. If D does not contain $\bar{l}$, then $D\,|\,\alpha = D\,|\,\alpha_l$ is contained in $F\,|\,\alpha$ and hence the claim follows immediately.

Assume now that $\bar{l} \in D$. Then, $D \in F_{\bar{l}}$ and thus $F \vdash_1 C \otimes_l D$. Since α falsifies all literals in C and since $C \otimes_l D$ is not a tautology, it follows that $F\,|\,\alpha \vdash_1 D \setminus \{\bar{l}\}$. Now, all the literals in $D \setminus \{\bar{l}\}$ that are not contained in $D\,|\,\alpha_l$ are anyhow falsified by α. Thus, propagating their negations does not change $F\,|\,\alpha$ and so $F\,|\,\alpha \vdash_1 D\,|\,\alpha_l$. It follows that C is literal-propagation redundant with respect to F. ☐

The LPR notion gives a simple proof why every non-empty RUP is LPR and thus a RAT:

Theorem 26. *If* C *is a non-empty* RUP *with respect to a formula* F*, then* C *is literal-propagation redundant with respect to* F.

Proof. Assume $C = (l_1 \vee \dots \vee l_n)$ is a RUP in F and let $\alpha = \bar{l}_1 \dots \bar{l}_n$ be the assignment precluded by C. As C is a RUP in F, we know that unit propagation derives a conflict on $F \wedge (\bar{l}_1) \wedge \dots \wedge (\bar{l}_n)$. But then unit propagation derives a conflict on $F|\alpha$. Hence, $F|\alpha \vdash_1 F|\alpha_l$ trivially holds for every $l \in C$ and thus C is literal-propagation redundant with respect to F. □

By allowing the witnessing assignments to disagree with α on more than only one literal, we obtain the more general notion of a set-propagation-redundant clause:

Definition 23. *Let F be a formula, C a clause, and α the assignment precluded by C. Then, C is* set-propagation redundant (SPR) *with respect to F if there exists a non-empty set $L \subseteq C$ of literals such that $F|\alpha \vdash_1 F|\alpha_L$.*

We denote the set $\{(F, C) \mid C$ is set-propagation redundant with respect to $F\}$ by SPR.

Example 18. *Consider the formula $F = (x \vee y) \wedge (x \vee \bar{y} \vee z) \wedge (\bar{x} \vee z) \wedge (\bar{x} \vee u) \wedge (\bar{u} \vee x)$ and the clause $(x \vee u)$. We can use the set $L = \{x, u\}$ to show that $(x \vee u)$ is set-propagation redundant with respect to F: First, note that C precludes the assignment $\alpha = \bar{x}\,\bar{u}$, and that $\alpha_L = x\,u$. Now, consider the formulas $F|\alpha = (y) \wedge (\bar{y} \vee z)$ and $F|\alpha_L = (z)$. Clearly, $F|\alpha \vdash_1 F|\alpha_L$ since unit propagation derives a conflict on $(y) \wedge (\bar{y} \vee z) \wedge (\bar{z})$. Hence, $(x \vee u)$ is set-propagation redundant with respect to F. Observe also that $(x \vee u)$ is not literal-propagation redundant with respect to F.*

In contrast to LPR and RAT, the redundancy property of set-propagation-redundant clauses (SPR) generalizes set-blocked clauses ($\mathsf{SET}_{\mathsf{BC}}$). To show this, we first characterize set-blocked clauses as follows:

Lemma 27. *Let F be a formula, C a clause, $L \subseteq C$ a non-empty set of literals, and α the assignment precluded by C. Then, C is set-blocked by L in F if and only if, for every $D \in F$, $D|\alpha = \top$ implies $D|\alpha_L = \top$.*

Proof. For the "only if" direction, assume that there exists a clause $D \in F$ such that $D|\alpha = \top$ but $D|\alpha_L \neq \top$. Then, since α and α_L disagree only on literals in L, it follows that D contains a literal $l \in \bar{L}$ and thus $D \in F_{\bar{L}}$. Now, α_L falsifies exactly the literals in $(C \setminus L) \cup \bar{L}$, and since α_L does not satisfy any literals of D, it follows that $D \notin F_L$, and that there exists no literal $l \in D$ such that its complement $\bar{l}$ is contained in $(C \setminus L)$. But then the set-resolvent $C \otimes_L D$ is not a tautology and so C is not set-blocked by L in F.

For the "if" direction, suppose C is not set-blocked by L in F, meaning that there exists a clause $D \in F_{\bar{L}} \setminus F_L$ such that the set-resolvent $C \otimes_L D = (C \setminus L) \cup (D \setminus \bar{L})$ is not a tautology. It follows that D does not contain any literals of L and that $D \setminus \bar{L}$ does not contain any literal $\bar{l}$ such that $l \in C \setminus L$. But, α_L falsifies exactly the literals in $(C \setminus L) \cup \bar{L}$, and thus α_L does not satisfy D. Now, since α falsifies L and since $D \in F_{\bar{L}}$, we know that $D|\alpha = \top$. Hence, $D|\alpha = \top$ does not imply $D|\alpha_L = \top$. □

We can now use this lemma to prove that set-propagation-redundant clauses generalize set-blocked clauses:

Theorem 28. *If a clause C is set-blocked in a formula F, it is set-propagation redundant with respect to F.*

Proof. Assume that C is set-blocked by a set L in F and let α be the assignment precluded by C. We show that $F|\alpha \supseteq F|\alpha_L$, which implies that $F|\alpha \vdash_1 F|\alpha_L$, and therefore that C is set-propagation redundant with respect to F. Let $D|\alpha_L \in F|\alpha_L$. First, note that D cannot be contained in F_L, for otherwise $D|\alpha_L = \top$ and thus $D|\alpha_L \notin F|\alpha_L$. Second, observe that D can also not be contained in $F_{\bar{L}}$, since that would imply that $D|\alpha = \top$ and thus, by Lemma 27, $D|\alpha_L = \top$. Therefore, $D \notin F_L \cup F_{\bar{L}}$ and so $D|\alpha = D|\alpha_L$. But then, $D|\alpha_L \in F|\alpha$. It follows that $F|\alpha \supseteq F|\alpha_L$. □

We thus know that set-propagation-redundant clauses generalize both RATs and set-blocked clauses. Actually, they are even a *strict* generalization since the redundancy properties of RATs and set-blocked clauses are incomparable (Theorem 23).

Note that $F|\alpha \vdash_1 F|\alpha_L$ is equivalent to $F|\alpha \vdash_1 F_{\bar{L}}|\alpha_L$. To see this, observe that if a clause $D|\alpha_L \in F|\alpha_L$ contains no literals from $\bar{L}$, then α_L does not assign any of its literals, in which case $F|\alpha \vdash_1 D|\alpha_L$ trivially holds since $D|\alpha_L$ is contained in $F|\alpha$. To check if a clause is set-propagation redundant, we therefore only need to check for each $D \in F_{\bar{L}}$, if $F|\alpha \vdash_1 D|\alpha$.

By giving full freedom to the witnessing assignments, i.e., by only requiring them to satisfy C, we finally arrive at the notion of a propagation-redundant clause:

Definition 24. *Let F be a formula, C a clause, and α the assignment precluded by C. Then, C is* propagation redundant (PR) *with respect to F if there exists an assignment ω such that ω satisfies C and $F|\alpha \vdash_1 F|\omega$.*

We denote the set $\{(F, C) \mid C$ is propagation redundant with respect to $F\}$ by PR.

Example 19. *Let $F = (x \lor y) \land (\bar{x} \lor y) \land (\bar{x} \lor z)$, $C = (x)$, and let $\omega = x\,z$ be the witnessing assignment. Then, C precludes the assignment $\alpha = \bar{x}$, and ω satisfies C. Now, consider the formulas $F|\alpha = (y)$ and $F|\omega = (y)$. Clearly, $F|\alpha \vdash_1 F|\omega$, and so C is propagation redundant with respect to F. Note that C is not set-propagation redundant with respect to F because for $L = \{x\}$, we have $\alpha_L = x$ and so $F|\alpha_L$ contains the two unit clauses (y) and (z), but it does not hold that $F|\alpha \vdash_1 (z)$. The fact that ω satisfies the literal z which is not contained in C is crucial for ensuring propagation redundancy.*

Deciding if a clause is propagation redundant with respect to a formula is NP-complete in general. To prove this, we define the corresponding decision problem:

Definition 25. *The* propagation-redundancy problem *is the following problem: Given a formula F and a clause C, decide if C is propagation redundant with respect to F.*

Theorem 29. *The propagation-redundancy problem is NP-complete.*

Proof. We show NP-membership followed by NP-hardness.

NP-membership: Let α be the assignment precluded by C. To decide whether or not C is propagation redundant with respect to F, just guess an assignment ω and check (in polynomial time) if $F|_\alpha \vdash_1 F|_\omega$.

NP-hardness: We give a polynomial reduction from the satisfiability problem of propositional logic. Let F be an input formula (in CNF) for the satisfiability problem. We define the following reduction function:

$$f(F) = (G, C),$$

where $C = (\bar{v})$ is a unit clause for some fresh variable v that does not occur in F, and G is obtained from F by adding to each clause the literal v. We show that F is satisfiable if and only if C is propagation redundant with respect to G.

For the "only if" direction, suppose F is satisfied by some assignment ω and let $\alpha = v$ be the assignment precluded by C. Now, define a new assignment ω' that agrees with ω on $var(\omega)$ but additionally falsifies v. Then, ω' disagrees with α on v. Moreover, since ω satisfies F, it satisfies G. Hence, ω' satisfies G and thus $G|_{\omega'} = \emptyset$, implying that $G|_\alpha \vdash_1 G|_{\omega'}$. It follows that C is propagation redundant with respect to G.

For the "if" direction, assume that C is propagation redundant with respect to G and let $\alpha = v$ be the assignment precluded by C. Then, there exists an assignment ω' such that $G|_\alpha \vdash_1 G|_{\omega'}$ and ω' falsifies C, meaning that $\omega'(v) = 0$. Since every clause in G contains v, it follows that α satisfies G and so it must be the case that ω' satisfies G. Since $\omega'(v) = 0$ and $G|_{\bar{v}} = F$, it follows that ω' satisfies F. ☐

Finally, the following example shows that PR does not generalize the redundancy property of super-blocked clauses ($\mathsf{SUP_{BC}}$):

Example 20. *Let $F = (e \lor \bar{x}) \land (\bar{e} \lor \bar{y})$ and let $C = (x \lor y)$. To see that C is super-blocked in F observe first that $ext_F(C)$, the set of external variables of C in F, is the set $\{e\}$. We need to show that C is set-blocked in $F|_e = (\bar{y})$ and in $F|_{\bar{e}} = (\bar{x})$. But this is trivial since $\bar{x}$ does not occur in $F|_e$, hence x is a pure literal in $F|_e$ and thus C is blocked by x in $F|_e$. Likewise, C is blocked by y in $F|_{\bar{e}}$. We conclude that C is super-blocked in F.*

In contrast, C is not propagation redundant with respect to F: Note that $F|_\alpha = \emptyset$ where $\alpha = \bar{x}\bar{y}$ is the assignment precluded by C. If C were propagation redundant with respect to F, there would exist an assignment ω such that $F|_\alpha \vdash_1 F|_\omega$. But this can only be the case if ω falsifies all clauses in F, which is impossible since ω cannot falsify both e and $\bar{e}$.

As we have already seen that there exist RAT clauses that are not super-blocked and since PR generalizes RAT, we conclude:

Theorem 30. PR *and* $\mathsf{SUP_{BC}}$ *are incomparable.*

2.2.4 Globally-Blocked Clauses

The difference between set-propagation-redundant clauses and propagation-redundant clauses is as follows: For a set-propagation-redundant clause, we allow the witnessing assignment α_L to differ from α (the assignment precluded by the clause) only on a set L of literals that are contained in the clause itself. In contrast, for propagation-redundant clauses, we allow the witnessing assignment ω to differ from α on arbitrary literals.

This brings us back to set-blocked clauses because we can generalize them by loosening their definition in a similar way. Remember that a clause C is set-blocked in a formula F if it contains a set $L \subseteq C$ of literals such that for every clause $D \in F_{\bar{L}} \setminus F_L$, the set-resolvent $C \otimes_L D$ is a tautology. By giving up the requirement that L be a subset of C—only requiring that L be a non-tautological set of literals (i.e., a set containing no complementary literals) that contains at least one literal of C—we arrive at *globally-blocked clauses*:

Definition 26. *A clause C is* globally blocked *in a formula F if there exists a non-tautological set L of literals such that $L \cap C \neq \emptyset$ and for every clause $D \in F_{\bar{L}} \setminus F_L$, the set-resolvent $C \otimes_L D$ is a tautology.*

We say that C is globally blocked *by* L in F, and we write $\mathsf{G_{BC}}$ to refer to the set $\{(F, C) \mid C$ is globally blocked in $F\}$. Note that if L were allowed to contain complementary literals, then every clause would be globally blocked by the set L of all literals, since in this case $F_{\bar{L}} \setminus F_L$ is empty.

Example 21. *Consider the formula $F = (\bar{x} \vee y) \wedge (\bar{y} \vee z) \wedge (x \vee \bar{z})$ and the clause $C = (\bar{x} \vee y)$. To see that C is globally blocked in F, consider the set $L = \{y, z\}$ and the formulas $F_{\bar{L}} = (\bar{y} \vee z) \wedge (x \vee \bar{z})$ and $F_L = (\bar{x} \vee y) \wedge (\bar{y} \vee z)$. We then have $F_{\bar{L}} \setminus F_L = (x \vee \bar{z})$, and since the set-resolvent $C \otimes_L (x \vee \bar{z}) = (\bar{x} \vee x)$ is a tautology, C is globally blocked in F. Note that C is not set-blocked in F.*

Since C in the above example is not set-blocked in F, we can conclude that globally-blocked clauses are a strict generalization of set-blocked clauses. In fact, if we are given a clause that is set-blocked by a non-empty set L, then we can remove from L all but one literal. The resulting clause is guaranteed to be globally blocked:

Theorem 31. *If a clause $(c_1 \vee \dots \vee c_m \vee l_1 \vee \dots \vee l_n)$ is set-blocked by $L = \{l_1, \dots, l_n\}$ in a formula F, then the clause $(c_1 \vee c_m \vee l_i)$ is globally blocked by L in F for $1 \leq i \leq n$.*

Proof. Suppose $C = (c_1 \vee \dots \vee c_m \vee l_1 \vee \dots \vee l_n)$ is set-blocked by $L = \{l_1, \dots, l_n\}$ in F, meaning that the set-resolvent $C \otimes_L D$ is a tautology for every clause $D \in F_{\bar{L}} \setminus F_L$. Now, observe that $C \setminus L = (c_1 \vee c_m \vee l_i) \setminus L$ for $1 \leq i \leq n$. Therefore, $C \otimes_L D = (C \setminus L) \cup (D \setminus \bar{L}) = (c_1 \vee c_m \vee l_i) \otimes_L D$ is a tautology for every clause $D \in F_{\bar{L}} \setminus F_L$. It follows that $(c_1 \vee \dots c_m \vee l_i)$ is globally blocked by L in F for $1 \leq i \leq n$. $\square$

The redundancy of globally-blocked clauses follows from the fact that they are propagation redundant:

Theorem 32. *If a clause C is globally blocked in a formula F, then C is propagation redundant with respect to F.*

Proof. Let C be a clause that is globally blocked in a formula F and let α be the assignment precluded by C. We have to show that there exists an assignment ω such that ω satisfies C and $F|\alpha \vdash_1 F|\omega$. Since C is globally blocked in F, there exists a set L of literals such that $L \cap C \neq \emptyset$ and for each clause $D \in F_{\bar{L}} \setminus F_L$, the set-resolvent $C \otimes_L D$ is a tautology. Now, define $\omega = \alpha_L$ and let $D|\omega$ in $F|\omega$. Since $L \cap C \neq \emptyset$, we know that ω satisfies C. Moreover, as $D|\omega$ is contained in $F|\omega$, we know that D is not satisfied by ω and thus $D \notin F_L$. Towards a contradiction, assume now that $D \in F_{\bar{L}}$. Then, the set-resolvent $C \otimes_L D$ is a tautology, which means that $D \setminus \bar{L}$ contains a literal that is satisfied by α and thus D must also be satisfied by ω. But then $D|\omega$ is not contained in $F|\omega$, a contradiction. Assume thus that $D \notin F_{\bar{L}}$. We then know that $D|\alpha = D|\omega$ and thus $D|\omega \in F|\alpha$. But then $F|\alpha \vdash_1 D|\omega$ trivially holds. It follows that C is propagation redundant with respect to F. □

Observe that by not requiring that L be a subset of C, we sometimes need to consider clauses outside the resolution neighborhood of C to check if C is globally blocked:

Example 22. *Consider the formula $F = (\bar{x} \vee y)$ and the unit clause (x). Clearly, (x) is not set-blocked in F since the only set that could set-block (x) is the set $\{x\}$, but the resolvent $(x) \otimes_x (\bar{x} \vee y) = (y)$ is not a tautology. However, (x) is* globally *blocked in F. To see this, let $L = \{x, y\}$ and observe that $F_{\bar{L}} \setminus F_L = \emptyset$. Now, consider the formula G, obtained from F by adding the clause $(\bar{y})$. Then, (x) has the same resolution neighborhood in both F and G. But, $G_{\bar{L}} \setminus G_L = (\bar{y})$, and the set-resolvent $(x) \otimes_L (\bar{y}) = \bot$ is not a tautology. Therefore, (x) is not globally blocked by L in G. It is easy to see that any other set of literals does also not globally block (x) in G and thus (x) is not globally blocked in G.*

In contrast to BC, $\mathsf{SET}_{\mathsf{BC}}$, and $\mathsf{SUP}_{\mathsf{BC}}$, the redundancy property of globally-blocked clauses is therefore not a local redundancy property. This leads to the landscape of redundancy properties in Figure 2.3. Next, we discuss relations of our redundancy properties with concepts from the literature before we use them to define proof systems.

2.3 Relation to Concepts From the Literature

Our new global redundancy properties are related to *variable instantiation* [ABCH02], *autarkies* [MS85], and *safe assignments* [WFS06].

If $F|\bar{l} \models F|l$ holds for some literal l, then *variable instantiation*, as described by Andersson et al. [ABCH02], says that F and $F|l$ are equisatisfiable. Analogously, our

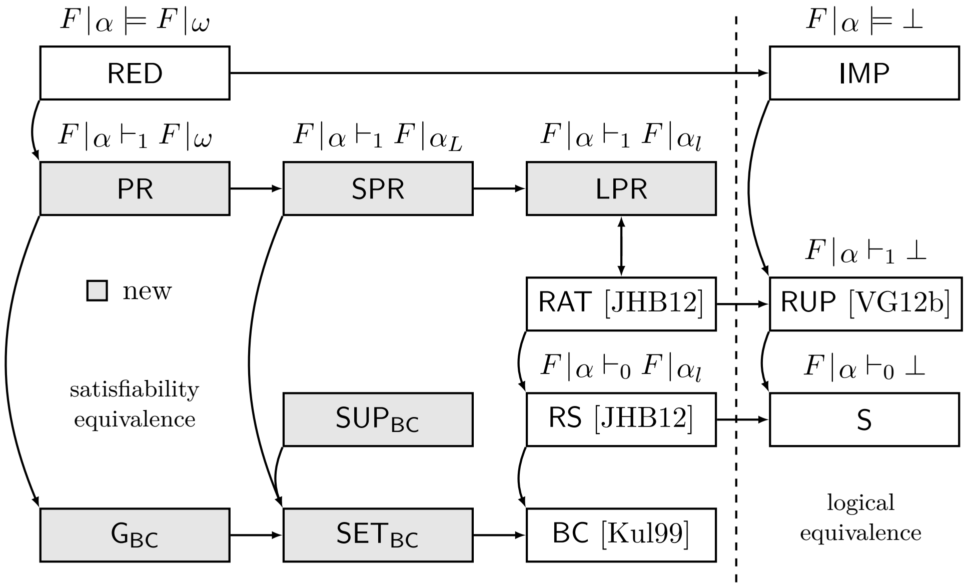


Figure 2.3: Final landscape of redundancy properties including globally-blocked clauses. A path from a redundancy property X to a redundancy property Y indicates that X is more general than Y.

implication-based redundancy characterization (Definition 24 on page 34) identifies the unit clause (l) as redundant with respect to F. Variable instantiation is thus a special case of Definition 24.

As discussed in Section 2.1.4, an assignment ω is an autarky [MS85] for a formula F if it satisfies all clauses of F that contain a literal to which ω assigns a truth value. Moreover, if an assignment ω is an autarky for a formula F, then F and $F|\omega$ are equisatisfiable. Similarly, propagation redundancy allows us to add all the unit clauses satisfied by an autarky, with the autarky serving as a witness: Let ω be an autarky for some formula F, let (l) be a unit clause for a literal l satisfied by ω, and let $\alpha = \bar{l}$ be the assignment precluded by C. Notice that $F|\alpha \supseteq F|\omega$ and thus (l) is propagation redundant with respect to F.

According to Weaver, Franco, and Schlipf [WFS06], an assignment ω is considered *safe* if, for every assignment α with $var(\alpha) = var(\omega)$, it holds that $F|\alpha \models F|\omega$. Weaver et al. showed that if an assignment ω is safe, then $F|\omega$ and F are equisatisfiable. In a similar fashion, our approach allows us to preclude all the assignments $\alpha \neq \omega$ by adding the corresponding clauses to F. Through this, we obtain a formula that is logically equivalent to $F|\omega$. Note that safe assignments generalize autarkies and variable instantiation. Moreover, while safe assignments only allow the application of an assignment ω to a formula F if $F|\alpha \models F|\omega$ holds for *all* assignments $\alpha \neq \omega$, our approach enables us to preclude an assignment α as soon as $F|\alpha \models F|\omega$.

CHAPTER 3

Proof Systems Based on Redundant Clauses

In this chapter, we deal with proof systems that are based on the addition of redundant clauses. Intuitively, proof systems describe methods for showing that formulas are unsatisfiable. Suppose you are asked if the following formula is satisfiable:

$$(x \vee y) \wedge (\bar{x} \vee z) \wedge (\bar{x} \vee \bar{z}) \wedge (z)$$

After inspecting the formula for a while, you conclude that it is satisfiable. How do you prove that the formula is indeed satisfiable? It's easy, you just find a satisfying assignment—for instance, the assignment $\bar{x}yz$—and demonstrate that your assignment satisfies at least one literal in each of the clauses.

Now we add the clause $(x \vee \bar{y})$. Same question as before: is it satisfiable?

$$(x \vee y) \wedge (\bar{x} \vee z) \wedge (\bar{x} \vee \bar{z}) \wedge (z) \wedge (x \vee \bar{y})$$

Again, you investigate the formula for a while just to find that this time the formula is unsatisfiable. But how are you going to prove this? Will you just go over all possible assignments and show that each of them falsifies a clause? What if, in a next step, you are presented with an unsatisfiable formula that has not only three but eight different variables? Will you go over all 256 possible assignments? Surely, there must be a more elegant way, and this is where proof systems come into play. Formally, we use the following notion of a proof system, which is due to Cook and Reckhow [CR79]:

Definition 27. *A* proof system *for propositional formulas in CNF is a surjective polynomial-time-computable function $f : \Sigma^* \to \mathcal{F}$ where Σ is some alphabet and $\mathcal{F}$ is the set of all unsatisfiable formulas.*

A proof system can thus be seen as a proof-checking function f that takes a *proof candidate* P (which is a string over Σ) together with an unsatisfiable formula F and checks in polynomial time if P is a correct proof of F. The requirement that f is surjective means that there must exist a proof for *every* unsatisfiable formula. We sometimes use the word *proof system* in a more colloquial way to denote the rules that define what constitutes a correct proof of a certain type. One example for a proof system is the well-known resolution proof system:

Definition 28. *The resolution proof system defines that a* resolution proof *of a formula F is a sequence $C_1, \dots, C_m$ of clauses such that $C_m = \bot$ and every clause C_i $(1 \leq i \leq m)$ is either contained in F or it is a resolvent of two previous clauses C_j, C_k $(j, k < i)$.*

Since every resolvent is implied by its premises, a valid proof can only derive $\bot$ if the original formula is unsatisfiable. Moreover, it can be shown that there exists a resolution proof for every unsatisfiable formula (see, e.g., [Lei97]).

The resolution proof system is captured by Definition 27 as follows: Define Σ as the set of symbols used to construct resolution proofs (including $\wedge$, $\vee$, $\bot$, propositional variables, etc.) and f as the function that maps every valid proof $C_1, \dots, C_m$ to the formula containing the clauses of $C_1, \dots, C_m$ that were not derived with the resolution rule. Finally, define $f(P) = \bot$ for each $P \in \Sigma^*$ that is not a valid resolution proof. As it can be easily checked if some $P \in \Sigma^*$ is a valid resolution proof, f is polynomial-time computable, and since there exists a resolution proof for every unsatisfiable formula, f is surjective.

In the rest of the thesis, to show that a proposed system is indeed a proof system according to Definition 27, we show that it is *sound* (i.e., if P is a proof of F, then F is unsatisfiable) and *complete* (there exists a proof for every unsatisfiable formula) and that the correctness of proofs can be checked in polynomial time.

As already mentioned, proof systems define ways to certify the unsatisfiability of formulas. If a proof system for propositional logic lends itself to automation, it can form the basis of a SAT solver, specifying what the solver can do to evaluate a formula. As we will see in more detail later, most state-of-the-art SAT solvers are based on the *resolution* proof system. Unfortunately, there exist only exponentially large resolution proofs for several seemingly easy problems [Hak85, Urq87], implying that resolution-based solvers require exponential time to solve these problems. Among them are the so-called *pigeon hole formulas*, which, according to Nordström [Nor15], represent "*arguably the single most studied combinatorial principle in all of proof complexity.*"

By extending the resolution proof system with a simple rule that allows the introduction of definitions over new variables, Tseitin turned it into an exponentially stronger proof system known as *extended resolution* [Tse68]:

Definition 29. *An* extended-resolution proof *of a formula F is a sequence $C_1, \dots, C_m$ of clauses such that $C_m = \bot$ and every clause C_i $(1 \leq i \leq m)$ is (1) contained in F, or*

(2) a resolvent of two previous clauses C_j, C_k $(j, k < i)$, or (3) added by an application of the extension rule*: The extension rule adds the clauses $(x \lor a)$, $(x \lor b)$, and $(\bar{x} \lor \bar{a} \lor \bar{b})$ where x is a new variable not occurring in previous clauses.*

Note that the clauses introduced by the extension rule are equivalent to a definition of the form $(x \leftrightarrow \bar{a} \lor \bar{b})$. Up to this day, there are no known exponential lower-bounds on the size of extended-resolution proofs and thus extended resolution is seen as one of the most powerful proof systems. The introduction of new variables, however, blows up the search space of possible proofs, and it is often unclear which definitions should be added to a proof. Automatically finding useful clauses with new variables is therefore hard in practice and resulted only in limited success in the past [AKS10, MHB13].

In the following, we present new proof systems that are highly expressive even when we disallow the introduction of new variables. We illustrate the strength of our strongest proof system by providing short clausal proofs of the pigeon hole formulas—without introducing new variables. The size of the proofs is linear in the size of the formulas and the new clauses added in the proofs contain at most two literals. In these proofs, we add propagation-redundant clauses that are similar in nature to symmetry-breaking predicates [CGLR96, DBBD16]. We compare our proofs with existing proofs of the pigeon hole formulas in the DRAT proof system and show that our new proofs are much smaller. To verify the correctness of the proofs, we used a toolchain involving a formally verified proof checker for LRAT proofs [HJKW17] (for details see Section 3.3). Finally, we also describe an algorithm for directly checking the correctness of proofs in our proof systems.

3.1 Clausal Proofs

Given a formula $F = C_1 \land \cdots \land C_m$, a *clausal derivation* of a clause C_n from F is a sequence $(C_{m+1}, \omega_{m+1}), \dots, (C_n, \omega_n)$ of pairs where C_i is a clause and ω_i, called the *witness*, is a string (for all $i > m$). Such a sequence gives rise to formulas $F_m, F_{m+1}, \dots, F_n$, where $F_i = C_1 \land \cdots \land C_i$. We call F_i the *accumulated formula* corresponding to the i-th proof step. A clausal derivation is *correct* if every clause C_i $(i > m)$ is redundant with respect to the formula F_{i-1} and if this redundancy can be checked in polynomial time (with respect to the size of the proof) using the witness ω_i. A clausal derivation is a (*refutation*) *proof* of a formula F if it derives the empty clause, i.e., if $C_n = \bot$. Clearly, since every clause-addition step preserves satisfiability, and since the empty clause is unsatisfiable, a refutation proof of F certifies the unsatisfiability of F. Note that the witnesses can also be empty, in which case a clausal derivation boils down to a simple sequence of clauses.

By specifying in detail what kind of redundant clauses—and corresponding witnesses—can be added to a clausal derivation, we obtain concrete proof systems. This is usually done by choosing an efficiently checkable redundancy property that guarantees that the addition of clauses fulfilling this property preserves unsatisfiability. A popular example for a clausal proof system is DRAT [WHHJ14], the de facto standard for unsatisfiability proofs in practical SAT solving. DRAT allows the addition of a clause if it is a RAT (see Definition 20). As it can be efficiently checked (even without using an explicit witness) if a clause is a RAT with respect to a formula, and since RATs cover many types of redundant clauses, the DRAT proof system is very powerful.

The strength of a clausal proof system depends on the generality of the underlying redundancy property—the more general the redundancy property, the more clauses we are allowed to add. A more general redundancy property thus gives us more freedom when using a specific proof system to prove the unsatisfiability of a formula.

We now explicitly define the PR proof system as an instance of a clausal proof system:

Definition 30. *Given a formula $F = C_1 \land \dots \land C_m$, a* PR derivation *of a clause C_n from F is a sequence $(C_{m+1}, \omega_{m+1}), \dots, (C_n, \omega_n)$ where for every pair (C_i, ω_i), one of the following holds: (1) ω_i is an assignment that satisfies C_i and $F_{i-1}\,|\,\alpha_i \vdash_1 F_{i-1}\,|\,\omega_i$ with α_i being the assignment precluded by C_i, or (2) $C_n = \bot$ and $F_{n-1} \vdash_1 \bot$. A* PR *derivation of $\bot$ from F is a* PR proof *of F.*

The proof systems LPR and SPR are defined accordingly. In the definition above, we treat the empty clause separately because only non-empty clauses can be propagation redundant. If we allow the mentioned proof systems to delete arbitrary clauses, we obtain the proof systems DLPR (which coincides with DRAT), DSPR, and DPR.

Note that if we wanted to stick strictly to Definition 27 of a proof system, then we would need to include the clauses of F into a proof. In practice, however, proofs and formulas are often treated separately, meaning that proof checkers expect the formula and the proof as separate inputs.

All our proof systems are sound because the clause additions in these systems preserve satisfiability and thus the empty clause can only be derived if the original formula is unsatisfiable. To see that the proof systems are complete, observe that every resolution proof is an LPR proof and thus also a proof in all our other proof systems: We have already seen that resolvents are RUPs and that non-empty RUPs are literal-propagation redundant (Theorem 26). Hence, resolvents are literal-propagation redundant and thus every resolution proof is an LPR proof.

Actually, every extended-resolution proof is also an LPR proof: Consider the extension rule of extended resolution, which adds the clauses $(x \lor a)$, $(x \lor b)$, and $(\bar{x} \lor \bar{a} \lor \bar{b})$, where x is a new variable. The LPR proof system allows us first to add the clauses $(x \lor a)$ and $(x \lor b)$ since there are no resolvents upon the new variable x and thus these clauses are actually blocked clauses, which are literal-propagation redundant. Finally, we can add the clause $(\bar{x} \lor \bar{a} \lor \bar{b})$ since the only resolvents of this clause upon $\bar{x}$ are the tautologies $(a \lor \bar{a} \lor \bar{b})$ and $(b \lor \bar{a} \lor \bar{b})$, obtained by resolving upon $\bar{x}$ with $(x \lor a)$ and $(x \lor b)$, respectively. Hence, $(\bar{x} \lor \bar{a} \lor \bar{b})$ is also a blocked clause.

Remember that a clause C is set-propagation redundant with respect to a formula F if it contains a set L of literals such that $F\,|\,\alpha \vdash_1 F\,|\,\alpha_L$, with α being the assignment precluded by C. Since L is a subset of C, we do not need to add the assignment α_L explicitly to an SPR proof. By requiring that L must consist of the first literals of C when adding C to a proof (viewing a clause as a sequence of literals), we can give α_L implicitly, and thereby ensure that the SPR property is efficiently decidable. For instance, when a proof contains the clause $(l_1 \vee \cdots \vee l_k)$, we first check if the SPR property holds under the assumption that $L = \{l_1\}$. If not, we proceed by assuming that $L = \{l_1, l_2\}$, and so on until $L = \{l_1, \ldots, l_k\}$. We thus need to check only linearly many candidate sets in the worst case. The same holds for the LPR proof system, where we can require the literal l (for which $F|\alpha \vdash_1 F|\alpha_l$ is supposed to hold) to appear as the first literal of the clause, which is actually the case for DRAT in practice.

In the following, we consider PR proofs *without* new variables.

3.2 Short PR Proofs of the Pigeon Hole Principle

In a landmark article, Haken [Hak85] showed that pigeon hole formulas cannot be refuted by resolution proofs that are of polynomial size with respect to the size of the formulas. In contrast, Cook [Coo76] proved that there are actually polynomial-size refutations of the pigeon hole formulas in the stronger proof system of extended resolution. What distinguishes extended resolution from general resolution is that it allows the introduction of new variables via definitions. Cook showed how such definitions can be used to reduce a pigeon hole formula of size n to a pigeon hole formula of size $n-1$ over new variables.

Since every extended-resolution proof is also a PR proof, the short proofs of Cook can also be obtained in the PR proof system as long as we allow the introduction of new variables. In the following, however, we illustrate how the PR proof system admits short proofs of pigeon hole formulas even without the introduction of new variables. This shows that the PR system is strictly stronger than the resolution calculus, even when we forbid the introduction of new variables. A pigeon hole formula PHP_n intuitively encodes that $n+1$ pigeons have to be assigned to n holes such that no hole contains more than one pigeon. In the encoding, a variable $x_{p,h}$ intuitively denotes that pigeon p is assigned to hole h:

$$PHP_n := \bigwedge_{1 \leq p \leq n+1} (x_{p,1} \vee \cdots \vee x_{p,n}) \wedge \bigwedge_{1 \leq p < q \leq n+1} \bigwedge_{1 \leq h \leq n} (\bar{x}_{p,h} \vee \bar{x}_{q,h})$$

The clauses in the first conjunction encode that every pigeon is assigned to at least one hole. The clauses in the second conjunction encode that no two pigeons are assigned to the same hole. Clearly, pigeon hole formulas are unsatisfiable. The main idea behind our approach is similar to that of Cook, namely to reduce a pigeon hole formula PHP_n to the smaller PHP_{n-1}. The difference is that in our case PHP_{n-1} is still defined on the same variables as PHP_n. Therefore, reducing PHP_n to PHP_{n-1} boils down to deriving the clauses $(x_{p,1} \vee \cdots \vee x_{p,n-1})$ for $1 \leq p \leq n$.

Following Haken [Hak85], we use array notation for clauses: Every clause is represented by an array of $n+1$ columns and n rows. An array contains a "+" in the p-th column and h-th row if and only if the literal $x_{p,h}$ occurs in the corresponding clause; the array contains a "$-$" in the p-th column and h-th row if and only if the literal $\bar{x}_{p,h}$ occurs in the corresponding clause. Representing PHP_n in array notation, we have for every clause $(x_{p,1} \vee \cdots \vee x_{p,n})$ an array in which the p-th column is filled with "+". Moreover, for every clause $(\bar{x}_{p,h} \vee \bar{x}_{q,h})$, we have an array that contains two "$-$" in row h—one in column p and the other in column q. For instance, PHP_3 is given in array notation as follows:

$$
\begin{array}{c|cccc|} & 1&2&3&4 \\ \cline{2-5} 1 & + & & & \\ 2 & + & & & \\ 3 & + & & & \\ \cline{2-5} \end{array}
\quad
\begin{array}{c|cccc|} & 1&2&3&4 \\ \cline{2-5} 1 & & + & & \\ 2 & & + & & \\ 3 & & + & & \\ \cline{2-5} \end{array}
\quad
\begin{array}{c|cccc|} & 1&2&3&4 \\ \cline{2-5} 1 & & & + & \\ 2 & & & + & \\ 3 & & & + & \\ \cline{2-5} \end{array}
\quad
\begin{array}{c|cccc|} & 1&2&3&4 \\ \cline{2-5} 1 & & & & + \\ 2 & & & & + \\ 3 & & & & + \\ \cline{2-5} \end{array}
$$

$$
\begin{array}{c|cccc|} & 1&2&3&4 \\ \cline{2-5} 1 & - & - & & \\ 2 & & & & \\ 3 & & & & \\ \cline{2-5} \end{array}
\quad
\begin{array}{c|cccc|} & 1&2&3&4 \\ \cline{2-5} 1 & - & & - & \\ 2 & & & & \\ 3 & & & & \\ \cline{2-5} \end{array}
\quad \cdots \quad
\begin{array}{c|cccc|} & 1&2&3&4 \\ \cline{2-5} 1 & & & & \\ 2 & & - & - & \\ 3 & & & & \\ \cline{2-5} \end{array}
\quad \cdots \quad
\begin{array}{c|cccc|} & 1&2&3&4 \\ \cline{2-5} 1 & & & & \\ 2 & & & & \\ 3 & & - & & - \\ \cline{2-5} \end{array}
\quad
\begin{array}{c|cccc|} & 1&2&3&4 \\ \cline{2-5} 1 & & & & \\ 2 & & & & \\ 3 & & & - & - \\ \cline{2-5} \end{array}
$$

We illustrate the general idea for reducing a pigeon hole formula PHP_n to the smaller PHP_{n-1} on the concrete formula PHP_3. It should, however, become clear from our explanation that the procedure works for every $n > 1$. If we want to reduce PHP_3 to PHP_2, we have to derive the following three clauses:

$$
\begin{array}{c|cccc|} & 1&2&3&4 \\ \cline{2-5} 1 & + & & & \\ 2 & + & & & \\ 3 & & & & \\ \cline{2-5} \end{array}
\quad
\begin{array}{c|cccc|} & 1&2&3&4 \\ \cline{2-5} 1 & & + & & \\ 2 & & + & & \\ 3 & & & & \\ \cline{2-5} \end{array}
\quad
\begin{array}{c|cccc|} & 1&2&3&4 \\ \cline{2-5} 1 & & & + & \\ 2 & & & + & \\ 3 & & & & \\ \cline{2-5} \end{array}
$$

We can do so by removing the "+" from the last row of every column full of "+", except for the last column, which can be ignored as it is not contained in PHP_2. The key observation is that a "+" in the last row of the p-th column can be removed with the help of so-called "diagonal clauses" of the form $(\bar{x}_{p,n} \vee \bar{x}_{n+1,h})$ $(1 \leq h \leq n-1)$. We are allowed to add these diagonal clauses since they are, as we will show, propagation redundant with respect to PHP_n. The arrays below represent the diagonal clauses introduced to remove the "+" from the last row of the first (left), second (middle), and third column (right):

$$
\begin{array}{c|cccc|} & 1&2&3&4 \\ \cline{2-5} 1 & & & & - \\ 2 & & & & \\ 3 & - & & & \\ \cline{2-5} \end{array}
\quad
\begin{array}{c|cccc|} & 1&2&3&4 \\ \cline{2-5} 1 & & & & \\ 2 & & & & - \\ 3 & - & & & \\ \cline{2-5} \end{array}
\qquad
\begin{array}{c|cccc|} & 1&2&3&4 \\ \cline{2-5} 1 & & & & - \\ 2 & & & & \\ 3 & & - & & \\ \cline{2-5} \end{array}
\quad
\begin{array}{c|cccc|} & 1&2&3&4 \\ \cline{2-5} 1 & & & & \\ 2 & & & & - \\ 3 & & - & & \\ \cline{2-5} \end{array}
\qquad
\begin{array}{c|cccc|} & 1&2&3&4 \\ \cline{2-5} 1 & & & & - \\ 2 & & & & \\ 3 & & & - & \\ \cline{2-5} \end{array}
\quad
\begin{array}{c|cccc|} & 1&2&3&4 \\ \cline{2-5} 1 & & & & \\ 2 & & & & - \\ 3 & & & - & \\ \cline{2-5} \end{array}
$$

We next show how exactly these diagonal clauses allow us to remove the bottom "+" from a column full of "+", or, in other words, how they help us to remove the literal $x_{p,n}$ from a clause $(x_{p,1} \vee \cdots \vee x_{p,n})$ $(1 \leq p \leq n)$. Consider, for instance, the clause $(x_{2,1} \vee x_{2,2} \vee x_{2,3})$ in PHP_3. Our aim is to remove the literal $x_{2,3}$ from this clause. Before we explain the procedure, we like to remark that proof systems based on propagation redundancy can easily simulate resolution: Since every resolvent of clauses in a formula F is implied by

F, the assignment α precluded by the resolvent must falsify F and thus $F|\alpha \vdash_1 \bot$. We explain our procedure textually before we illustrate it in array notation:

First, we add the diagonal clauses $D_1 = (\bar{x}_{2,3} \vee \bar{x}_{4,1})$ and $D_2 = (\bar{x}_{2,3} \vee \bar{x}_{4,2})$ to PHP_3. Now, we can derive the unit clause $(\bar{x}_{2,3})$ by resolving the two diagonal clauses D_1 and D_2 with the original pigeon hole clauses $P_1 = (\bar{x}_{2,3} \vee \bar{x}_{4,3})$ and $P_2 = (x_{4,1} \vee x_{4,2} \vee x_{4,3})$ as follows: We obtain $(\bar{x}_{2,3} \vee x_{4,2} \vee x_{4,3})$ by resolving D_1 with P_2. Then, we resolve this clause with D_2 to obtain $(\bar{x}_{2,3} \vee x_{4,3})$, which we resolve with P_1 to obtain $(\bar{x}_{2,3})$. Note that our proof system actually allows us to add $(\bar{x}_{2,3})$ immediately without carrying out all the resolution steps explicitly. Finally, we resolve $(\bar{x}_{2,3})$ with $(x_{2,1} \vee x_{2,2} \vee x_{2,3})$ to obtain the desired clause $(x_{2,1} \vee x_{2,2})$.

We next illustrate this procedure in array notation. We start by visualizing the clauses D_1, D_2, P_1, and P_2 that can be resolved to yield the clause $(\bar{x}_{2,3})$. The clauses are given in array notation as follows:

$$\begin{array}{c|cccc|} & 1 & 2 & 3 & 4 \\ \hline 1 & & & & - \\ 2 & & & & \\ 3 & & - & & \\ \hline \end{array} \quad \begin{array}{c|cccc|} & 1 & 2 & 3 & 4 \\ \hline 1 & & & & \\ 2 & & & & - \\ 3 & & - & & \\ \hline \end{array} \quad \begin{array}{c|cccc|} & 1 & 2 & 3 & 4 \\ \hline 1 & & & & \\ 2 & & & & \\ 3 & & - & & - \\ \hline \end{array} \quad \begin{array}{c|cccc|} & 1 & 2 & 3 & 4 \\ \hline 1 & & & & + \\ 2 & & & & + \\ 3 & & & & + \\ \hline \end{array} \qquad \begin{array}{c|cccc|} & 1 & 2 & 3 & 4 \\ \hline 1 & & & & \\ 2 & & & & \\ 3 & & - & & \\ \hline \end{array}$$

$$D_1 \qquad D_2 \qquad P_1 \qquad P_2 \qquad \bar{x}_{2,3}$$

We can then resolve $(\bar{x}_{2,3})$ with $(x_{2,1} \vee x_{2,2} \vee x_{2,3})$ to obtain $(x_{2,1} \vee x_{2,2})$:

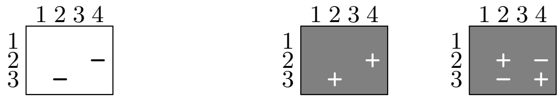
$$\begin{array}{c|cccc|} & 1 & 2 & 3 & 4 \\ \hline 1 & & & & \\ 2 & & & & \\ 3 & & - & & \\ \hline \end{array} \quad \begin{array}{c|cccc|} & 1 & 2 & 3 & 4 \\ \hline 1 & & + & & \\ 2 & & + & & \\ 3 & & + & & \\ \hline \end{array} \qquad \begin{array}{c|cccc|} & 1 & 2 & 3 & 4 \\ \hline 1 & & + & & \\ 2 & & + & & \\ 3 & & & & \\ \hline \end{array}$$

$$(\bar{x}_{2,3}) \qquad (x_{2,1} \vee x_{2,2} \vee x_{2,3}) \qquad (x_{2,1} \vee x_{2,2})$$

This illustrates how a clause of the form $(x_{p,1} \vee \cdots \vee x_{p,n})$ $(1 \leq p \leq n)$ can be reduced to a clause $(x_{p,1} \vee \cdots \vee x_{p,n-1})$. By repeating this procedure for every column p with $1 \leq p \leq n$, we can thus reduce a pigeon hole formula PHP_n to a pigeon hole formula PHP_{n-1} without introducing new variables. Note that the last step, in which we resolve the derived unit clause $(\bar{x}_{2,3})$ with the clause $(x_{2,1} \vee x_{2,2} \vee x_{2,3})$, is actually not necessary for a valid PR proof of a pigeon hole formula, but we added it to simplify the presentation.

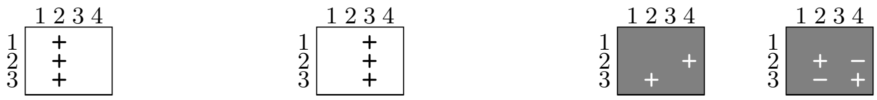
It remains to show that the diagonal clauses are indeed propagation redundant. To do so, we show that for every assignment $\alpha = x_{p,n}\, x_{n+1,h}$ that is precluded by a diagonal clause $(\bar{x}_{p,n} \vee \bar{x}_{n+1,h})$, it holds that for the assignment $\omega = \bar{x}_{p,n}\, \bar{x}_{n+1,h}\, x_{p,h}\, x_{n+1,n}$, $PHP_n|\alpha = PHP_n|\omega$, implying that $PHP_n|\alpha \vdash_1 PHP_n|\omega$. We also argue why other diagonal and unit clauses can be ignored when checking whether a new diagonal clause is propagation redundant.

We again illustrate the idea on PHP_3. We now use array notation also for assignments, i.e., a "+" ("–") in column p and row h denotes that the assignment makes variable $x_{p,h}$ true (false, respectively). Consider, for instance, the diagonal clause $D_2 = (\bar{x}_{2,3} \vee \bar{x}_{4,2})$ that precludes $\alpha = x_{2,3}\, x_{4,2}$. The corresponding witnessing assignment $\omega = \bar{x}_{2,3}\, \bar{x}_{4,2}\, x_{2,2}\, x_{4,3}$ can be seen as a "rectangle" with two "−" in the corners of one diagonal and two "+" in the other corners:



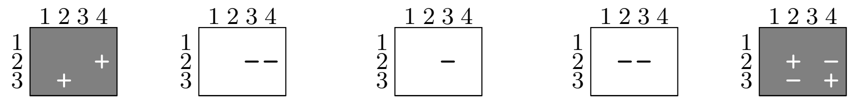
D_2 $\qquad$ α $\qquad$ ω

To see that $PHP_3\,|\,\alpha$ and $PHP_3\,|\,\omega$ coincide on clauses $(x_{p,1} \vee \cdots \vee x_{p,n})$, consider that whenever α and ω assign a variable of such a clause, they both satisfy the clause (since they both have a "+" in every column in which they assign a variable) and so they both remove it from PHP_3. For instance, in the following example, both α and ω satisfy $(x_{2,1} \vee x_{2,2} \vee x_{2,3})$ while both do not assign a variable of the clause $(x_{3,1} \vee x_{3,2} \vee x_{3,3})$:



$(x_{2,1} \vee x_{2,2} \vee x_{2,3})$ $\qquad$ $(x_{3,1} \vee x_{3,2} \vee x_{3,3})$ $\qquad$ α $\qquad$ ω

To see that $PHP_3\,|\,\alpha$ and $PHP_3\,|\,\omega$ coincide on clauses of the form $(\bar{x}_{p,h} \vee \bar{x}_{q,h})$, consider the following: If α falsifies a literal of $(\bar{x}_{p,h} \vee \bar{x}_{q,h})$, then the resulting clause is a unit clause for which one of the two literals is not assigned by α (since α does not assign two variables in the same row). Now, one can show that the same unit clause is also contained in $PHP_3\,|\,\omega$, where it is obtained from another clause: Consider, for example, again the assignment $\alpha = x_{2,3}\, x_{4,2}$ and the corresponding witnessing assignment $\omega = \bar{x}_{2,3}\, \bar{x}_{4,2}\, x_{2,2}\, x_{4,3}$ from above. The assignment α turns the clause $C = (\bar{x}_{3,2} \vee \bar{x}_{4,2})$ into the unit clause $C\,|\,\alpha = (\bar{x}_{3,2})$. The same clause is contained in $PHP_3\,|\,\omega$, as it is obtained from $C' = (\bar{x}_{2,2} \vee \bar{x}_{3,2})$ since $C'\,|\,\omega = C\,|\,\alpha = (\bar{x}_{3,2})$:



α $\qquad$ C $\qquad$ $C\,|\,\alpha = C'\,|\,\omega$ $\qquad$ C' $\qquad$ ω

Note that diagonal clauses and unit clauses that have been derived earlier can be ignored when checking whether the current one is propagation redundant. For instance, assume we are currently reducing PHP_n to PHP_{n-1}. Then, the assignments α and ω under consideration only assign variables in PHP_n. In contrast, the unit and diagonal clauses used for reducing PHP_{n+1} to PHP_n (or earlier ones) are only defined on variables outside of PHP_n. They are therefore contained in both $PHP_n\,|\,\alpha$ and $PHP_n\,|\,\omega$. We can also ignore earlier unit and diagonal clauses over variables in PHP_n, i.e., clauses used for

CNF Formula	DIMACS File	PR Proof File	Lemmas
	`p cnf 12 22`	**-3 -10** -3 -10 1 12 0	$\bar{x}_{1,3} \vee \bar{x}_{4,1}$
$x_{1,1} \vee x_{1,2} \vee x_{1,3}$	`1 2 3 0`	**-3 -11** -3 -11 2 12 0	$\bar{x}_{1,3} \vee \bar{x}_{4,2}$
$x_{2,1} \vee x_{2,2} \vee x_{2,3}$	`4 5 6 0`	**-3** 0	$\bar{x}_{1,3}$
$x_{3,1} \vee x_{3,2} \vee x_{3,3}$	`7 8 9 0`	**-6 -10** -6 -10 4 12 0	$\bar{x}_{2,3} \vee \bar{x}_{4,1}$
$x_{4,1} \vee x_{4,2} \vee x_{4,3}$	`10 11 12 0`	**-6 -11** -6 -11 5 12 0	$\bar{x}_{2,3} \vee \bar{x}_{4,2}$
$\bar{x}_{1,1} \vee \bar{x}_{2,1}$	`-1 -4 0`	**-6** 0	$\bar{x}_{2,3}$
$\bar{x}_{1,2} \vee \bar{x}_{2,2}$	`-2 -5 0`	**-9 -10** -9 -10 7 12 0	$\bar{x}_{3,3} \vee \bar{x}_{4,1}$
$\bar{x}_{1,3} \vee \bar{x}_{2,3}$	`-3 -6 0`	**-9 -11** -9 -11 8 12 0	$\bar{x}_{3,3} \vee \bar{x}_{4,2}$
$\bar{x}_{1,1} \vee \bar{x}_{3,1}$	`-1 -7 0`	**-9** 0	$\bar{x}_{3,3}$
$\bar{x}_{1,2} \vee \bar{x}_{3,2}$	`-2 -8 0`	**-2** 0	$\bar{x}_{1,2}$
$\bar{x}_{1,3} \vee \bar{x}_{3,3}$	`-3 -9 0`	**-5** 0	$\bar{x}_{2,2}$
...	...	0	$\perp$

Figure 3.1: Left, ten clauses of PHP_3 using the notation as elsewhere in this thesis and next to it the equivalent representation of these clauses in the DIMACS format used by SAT solvers. Right, the full PR refutation consisting of clause-witness pairs. A repetition of the first literal indicates the start of the optional witness.

reducing an earlier column or other diagonal clauses for the current column: If α assigns one of their variables, then ω satisfies them and so they are not in $PHP_n|\omega$.

To compare our PR proofs of the pigeon hole formulas with existing DRAT proofs and to verify their correctness (see Section 3.3), we wrote a script that generates the proofs automatically. The format of our PR proofs is an extension of the DRAT format: the first numbers of the i-th line denote the literals in the clause C_i. Positive numbers refer to positive literals, and negative numbers refer to negative literals. In case a witness ω_i is provided, the first literal in the clause is repeated to denote the start of the witness. As the witness needs to satisfy the clause, it is guaranteed to have a literal in common with the clause. Our format requires that such a literal occurs at the first position of the clause and of the witness. A 0 marks the end of a line. Figure 3.1 shows the formula and the PR proof of our running example PHP_3 from the previous section.

Table 3.1 compares our PR proofs with existing DRAT proofs of the pigeon hole formulas (hole*.cnf). It also compares PR proofs with existing DRAT proofs of formulas from another challenging benchmark suite of the SAT competition that allows two pigeons per hole (tph*.cnf). For the latter formulas, PR proofs can be constructed in a similar way as for the classical pigeon hole formulas. Notice that the PR proofs do not introduce new variables and that they contain fewer clauses than their corresponding formulas. The DRAT proof of PHP_n contains a copy of the formula PHP_k for each $k < n$.

Finally, we want to mention that short SPR proofs (without new variables) of the pigeon hole formulas can be constructed by first adding set-propagation-redundant clauses of the form $(\bar{x}_{p,n} \vee \bar{x}_{n+1,h} \vee x_{p,h} \vee x_{n+1,n})$ and then deriving diagonal clauses from them via resolution. We left these proofs out since they are twice as large as the PR proofs and their explanation is less intuitive.

Table 3.1: The sizes (in terms of variables and clauses) of pigeon hole formulas (hole*.cnf) and two-pigeons-per-hole formulas (tph*.cnf) as well as the sizes of their PR proofs (as described in Section 3.2) and their DRAT proofs (based on symmetry breaking [HHJW15]).

	Input Formula		PR Proof		DRAT Proof	
Formula	Variables	Clauses	Variables	Clauses	Variables	Clauses
hole10.cnf	110	561	110	385	440	3 685
hole11.cnf	132	738	132	506	572	5 236
hole12.cnf	156	949	156	650	728	7 228
hole13.cnf	182	1 197	182	819	910	9 737
hole20.cnf	420	4 221	420	2 870	3 080	49 420
hole30.cnf	930	13 981	930	9 455	9 920	234 205
hole40.cnf	1 640	32 841	1 640	22 140	22 960	715 040
hole50.cnf	2 550	63 801	2 550	42 925	44 200	1 708 925
tph8.cnf	136	5 457	136	680	3 520	834 963
tph12.cnf	300	27 625	300	2 300	11 376	28 183 301
tph16.cnf	528	87 329	528	5 456	not available, too large	
tph20.cnf	820	213 241	820	10 660	not available, too large	

PRcheck (formula $F_m = C_1, \dots, C_m$; PR proof $(C_{m+1}, \omega_{m+1}), \dots, (C_n, \omega_n)$)
for $i \in \{m+1, \dots, n\}$ **do**
  for $D \in F_{i-1}$ **do**
    if $D|\omega_i \neq \top$ **and** $(D|\alpha_i = \top$ **or** $D|\omega_i \subset D|\alpha_i)$ **then**
      if $F_{i-1}|\alpha_i \nvdash_1 D|\omega_i$ **then return** *failure*
  $F_i := F_{i-1} \cup \{C_i\}$
return *success*

Figure 3.2: Pseudo Code of the PR-Proof Checking Algorithm.

3.3 Checking the Correctness of PR proofs

We present two different approaches to checking the correctness of PR proofs. The first approach involves a chain of translations and a formally verified proof checker: We start with a PR proof and translate it into a DRAT proof using the recently invented tool `pr2drat` by Marijn Heule and Armin Biere [HB18]. We then use the `DRAT-trim` checker [WHHJ14] to optimize the DRAT proof (i.e., to remove redundant proof parts) and to convert it into the LRAT format. Finally, we check the correctness of the resulting LRAT proof using a formally verified proof checker [HJKW17]. We used this approach to certify the correctness of the proofs for the pigeon hole formulas presented in the previous section.

The second approach, which was implemented by Marijn Heule on top of his proof checker `DRAT-trim`, is to directly check PR proofs. Figure 3.2 shows the pseudo code of an algorithm for checking the correctness of PR proofs. Note that in the proof system DPR, we allow the deletion of *arbitrary* clauses. Because of this, nothing has to be checked for deletion steps and so the algorithm can be easily generalized to DPR proof checking. The first "if" statement is not necessary but significantly improves the efficiency of the algorithm.

The worst-case complexity of the algorithm is $\mathcal{O}(kn^3)$, where n is the size of the final formula and k is the maximal clause length. The reason for this is that there are $n - m$ iterations of the outer for-loop and for each of these iterations, the inner for-loop is performed $|F_i|$ times, i.e., once for every clause in F_i. Given that F_i contains i clauses, we know that the size of F is bounded by n. It follows that the inner for-loop is performed $\mathcal{O}(mn)$ times. Now, there is a unit-propagation test in the inner if-statement: If k is the maximal clause size and n is an upper bound for the size of the formula, then the complexity of unit propagation is known to be $\mathcal{O}(kn)$. Hence, the overall worst-case complexity of the algorithm is $\mathcal{O}(mkn^2) = \mathcal{O}(kn^3)$.

This complexity is the same as for RAT-proof checking and thus also for DRAT-proof checking. In fact, the pseudo-code for RAT-proof checking and PR-proof checking is the same apart from the first if-statement, which is always true in the worst case, both for RAT and PR. Although the theoretical worst-case complexity makes proof checking seem very expensive, it can be done quite efficiently in practice: For the DRAT proofs produced by solvers in the SAT competitions, we observed that the runtime of proof checking is close to linear with respect to the sizes of the proofs.

Finally, we want to highlight that verifying the PR property of a clause is relatively easy as long as a witnessing assignment is given. For an arbitrary clause *without* a witnessing assignment, however, deciding the PR property is an NP-complete problem (Theorem 29 on page 40). We therefore believe that in general, verifying a PR proof of a formula is simpler than the actual solving/proving.

Next, we use the PR proof system to define a new SAT solving paradigm.

CHAPTER 4

Satisfaction-Driven Clause Learning

Most state-of-the-art SAT solvers are based on the *conflict-driven clause learning* (CDCL) paradigm [MSS99, MMZ+01]. At its core, CDCL is based on the resolution proof system, which means that the same limitations that apply to resolution also apply to CDCL. Most importantly, a CDCL-based solver needs exponential time to solve formulas that have only resolution proofs of exponential size, such as the pigeon hole formulas.

To break this exponential barrier, we introduce *satisfaction-driven clause learning* (SDCL), a SAT solving paradigm that extends CDCL in such a way that it can exploit the strengths of our PR proof system. Intuitively, SDCL extends CDCL by pruning the search space of truth assignments more aggressively. While a pure CDCL solver learns only clauses that can be efficiently derived via resolution, an SDCL solver also learns stronger clauses. To learn these clauses, the solver uses so-called *pruning predicates*: Given a formula and an assignment (in practice, this is the assignment currently explored by the solver), a pruning predicate is a simple propositional formula that encodes the question if the assignment can be safely pruned from the search space. To perform the pruning, the solver learns the clause that precludes the assignment (see Definition 21 on page 34). Thus, while solving a single hard formula, SDCL solves several simple formulas to improve overall efficiency. Figure 4.1 illustrates how learned clauses can prune the search space.

In the following, we first discuss the conflict-driven clause learning paradigm. We then introduce satisfaction-driven clause learning and present two different pruning predicates. Finally, we present an experimental evaluation of SDCL on hard formulas.

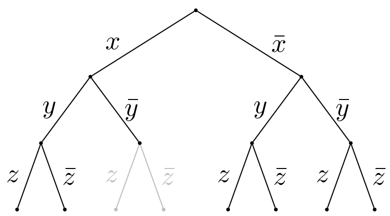


Figure 4.1: The tree represents the search space of truth assignments over the variables x, y, and z. Every branch from the root node to a leaf corresponds to an assignment. By learning the clause $(\bar{x} \vee y)$, a solver can prune all branches where x is true and y is false.

4.1 Conflict-Driven Clause Learning

Figure 4.2 shows the pseudo code of CDCL. In a nutshell, a CDCL solver performs the following operations to decide the satisfiability of a formula (for a more detailed discussion of CDCL, we refer to [MSLM09]):

First, the solver performs unit propagation until either it derives a conflict or the formula contains no more unit clauses. If it derives a conflict, it analyzes the conflict to learn a clause that prevents it from repeating similar (bad) decisions in the future ("clause learning"). In case this clause is the (unsatisfiable) empty clause, the solver can conclude that the formula is unsatisfiable. In case it is not the empty clause, the solver revokes some of its variable assignments ("backjumping") and then repeats the whole procedure again by performing unit propagation. If, however, the solver does not derive a conflict, there are two options: Either all variables are assigned, in which case the solver can conclude that the formula is satisfiable, or there are still unassigned variables, in which case the solver first assigns a truth value to an unassigned variable (the actual variable

```
CDCL(formula F)
α := ∅
forever do
   α := UnitPropagate(F, α)
   if α falsifies a clause in F then
      C := AnalyzeConflict()
      F := F ∧ C
      if C is the empty clause ⊥ then return UNSAT
      α := BackJump(C, α)
   else
      if all variables are assigned then return SAT
      l := Decide()
      α := α ∪ {l}
```

Figure 4.2: CDCL Algorithm.

and the truth value are chosen based on a so-called *decision heuristic*) and then continues by again performing unit propagation.

An important feature of CDCL is that a solver can only learn new clauses that are efficiently derivable from previous clauses via resolution. While this has the advantage that CDCL solvers can produce resolution proofs, it brings with it all the inefficiencies of the resolution proof system. We thus generalize CDCL in the following to allow the addition of even stronger clauses.

4.2 Generalizing Conflict-Driven Clause Learning

Our *satisfaction-driven clause learning* (SDCL) paradigm extends the CDCL paradigm in the following way: Whenever a CDCL solver finishes unit propagation without having derived a conflict and without having assigned all variables, it picks an unassigned variable and assigns a truth value to it. In contrast, an SDCL solver does not immediately after unit propagation make a new variable assignment. Instead, it first checks if the current assignment (and all its extensions) can be *pruned* from the search space by learning the clause that precludes the assignment. If the pruning cannot be performed, the SDCL solver makes a new variable assignment, just like a CDCL solver. If the pruning can be performed, however, it analyzes the clause that precludes the current assignment and possibly shortens it before adding it to the formula. It then revokes some of its variable assignments and continues by again performing unit propagation.

To test if the current assignment can be pruned, the solver generates a (possibly) simple formula and passes it to another SAT solver. An SDCL solver thus solves several simple formulas in order to solve a single hard formula. We call the simple formulas *pruning predicates*:

Definition 31. *Let F be a formula, α an assignment, and C the clause that precludes α. A* pruning predicate *for F and α is a formula $P_\alpha(F)$ such that the following holds: If $P_\alpha(F)$ is satisfiable, then C is redundant with respect to F.*

If the pruning predicate for a formula F and an assignment α is satisfiable, we can add the clause that precludes α to F without affecting satisfiability. As we will see, we can often learn a subclause of the clause that precludes α. The pseudo code for the SDCL paradigm is given in Figure 4.3. Removing the lines 9 to 12 would result in the classical CDCL algorithm. Line 9 corresponds to a solver call. The call of *AnalyzeWitness* in line 10 checks if the clause that precludes the current assignment can be shortened.

Next, we introduce two different pruning predicates and show how the resulting clauses can be shortened. When defining pruning predicates, we have to deal with an important trade-off: Solving them should be efficient to ensure usefulness in practice while they should be as satisfiable as possible to maximize pruning.

SDCL(formula F)

$\alpha := \emptyset$

forever do

  $\alpha := \mathit{UnitPropagate}(F, \alpha)$

  if α falsifies a clause in F **then**

    $C := \mathit{AnalyzeConflict}()$

    $F := F \wedge C$

    if C is the empty clause $\perp$ **then return** UNSAT

    $\alpha := \mathit{BackJump}(C, \alpha)$

  else if the pruning predicate $P_\alpha(F)$ is satisfiable **then**

    $C := \mathit{AnalyzeWitness}()$

    $F := F \wedge C$

    $\alpha := \mathit{BackJump}(C, \alpha)$

  else

    if all variables are assigned **then return** SAT

    $l := \mathit{Decide}()$

    $\alpha := \alpha \cup \{l\}$

Figure 4.3: SDCL Algorithm.

4.3 Pruning Predicates

We present two pruning predicates. We call them

- *positive reduct* and
- *filtered positive reduct.*

The two reducts differ in their generality. Given a formula F and an assignment α, the positive reduct is satisfiable if and only if the clause C that precludes α is set-blocked in F; the filtered positive reduct is satisfiable if and only if C is set-propagation redundant with respect to F. The positive reduct is therefore less satisfiable than the filtered positive reduct, but it is also easier to construct.

We start with the positive reduct, which is obtained from satisfied clauses of the original formula by removing unassigned literals. In the following, given a clause C and an assignment α, we denote by $\mathsf{touched}_\alpha(C)$ the subclause of C that contains exactly the literals assigned by α. Analogously, we denote by $\mathsf{untouched}_\alpha(C)$ the subclause of C that contains the literals *not* assigned by α.

Definition 32. *Given a formula F and an assignment α, the* positive reduct $\mathsf{p}_\alpha(F)$ *of F and α is the formula $G \wedge C$ where $G = \{\mathsf{touched}_\alpha(D) \mid D \in F \text{ and } D|_\alpha = \top\}$ and C is the clause that precludes α.*

We next show that the positive reduct is satisfiable if and only if the clause precluded by α is a set-blocked clause, which implies that it is also a propagation-redundant clause (remember that deciding set-blockedness and propagation redundancy are both NP-complete problems). We show later that we can usually shorten this set-blocked clause and thereby turn it into a PR clause that might not be set-blocked anymore.

Theorem 33. *Let F be a formula, α an assignment, and C the clause that precludes α. Then, C is set-blocked by L in F if and only if α_L satisfies the positive reduct $\mathsf{p}_\alpha(F)$.*

Proof. For the "only if" direction, assume that C is set-blocked by L in F, meaning that for every clause $D \in F_{\bar{L}} \setminus F_L$, the set-resolvent $C \otimes_L D$ is a tautology. We show that α_L satisfies $\mathsf{p}_\alpha(F)$. Clearly, α_L satisfies C since L is a non-empty subset of C. Now, let $D' \in \mathsf{p}_\alpha(F)$ be a clause that is different from C. Then, $D' = \mathsf{touched}_\alpha(D)$ for some clause $D \in F$. If $D \in F_L$, then D' is clearly satisfied by α_L. Moreover, if $D \notin F_{\bar{L}}$, then α agrees with α_L on $var(D)$ (and thus on $var(D')$), and since α satisfies $\mathsf{p}_\alpha(F)$, it follows that α_L satisfies D'. Assume now that $D \in F_{\bar{L}} \setminus F_L$. Then, the set-resolvent $C \otimes_L D$ is a tautology. This means that $D \setminus \bar{L}$ contains a literal $\bar{c}$ such that $c \in C \setminus L$. Since α falsifies C and since α_L agrees with α on $var(C \setminus L)$, we can conclude that α_L satisfies D'. It follows that α_L satisfies the positive reduct $\mathsf{p}_\alpha(F)$.

For the "if" direction, assume that α_L satisfies $\mathsf{p}_\alpha(F)$. We show that C is set-blocked by L in F. Let $D \in F_{\bar{L}} \setminus F_L$. Since α falsifies C, it falsifies L. Therefore, α satisfies $\bar{L}$ and thus $\mathsf{p}_\alpha(F)$ contains the clause $\mathsf{touched}_\alpha(D)$, obtained from a clause $D \in F$ by removing all literals that are not assigned by α. By assumption, α_L satisfies $\mathsf{touched}_\alpha(D)$ and since it falsifies $\bar{L}$, it must satisfy some literal $l \in \mathsf{touched}_\alpha(D) \setminus \bar{L}$. But then $\bar{l} \in C \setminus L$ and thus the set-resolvent $C \otimes_L D$ is a tautology. □

When constructing the positive reduct, we take all clauses of F that are satisfied by α and then remove from these clauses the literals that are not touched by α. In the *filtered positive reduct*, which we present next, we do not take *all* satisfied clauses of F but only those for which the untouched part is not implied by $F|\alpha$ via unit propagation:

Definition 33. *Let F be a formula and α an assignment. The* filtered positive reduct *$\mathsf{f}_\alpha(F)$ of F and α is the formula $G \wedge C$ where C is the clause that precludes α and $G = \{\mathsf{touched}_\alpha(D) \mid D \in F$ and $F|\alpha \nvdash_1 \mathsf{untouched}_\alpha(D)\}$.*

The filtered positive reduct is a subset of the positive reduct since $F|\alpha \nvdash_1 \mathsf{untouched}_\alpha(D)$ implies $D|\alpha = \top$. To see this, suppose $D|\alpha \neq \top$. Then, $D|\alpha$ is contained in $F|\alpha$ and since $\mathsf{untouched}_\alpha(D) = D|\alpha$, it follows that $F|\alpha \vdash_1 \mathsf{untouched}_\alpha(D)$. Therefore, the filtered positive reduct is obtained from the positive reduct by removing ("filtering out") every clause $\mathsf{touched}_\alpha(D)$ for which $F|\alpha \vdash_1 \mathsf{untouched}_\alpha(D)$.

Example 23. *Let $F = (x \vee y) \wedge (\bar{x} \vee y)$ and consider the assignment $\alpha = x$. The positive reduct $\mathsf{p}_\alpha(F) = (x) \wedge (\bar{x})$ is unsatisfiable whereas the filtered positive reduct $\mathsf{f}_\alpha(F) = (\bar{x})$, obtained by filtering out the clause (x), is satisfiable. The clause (x) is not contained in* *the filtered positive reduct because* $\mathsf{untouched}_\alpha(x \vee y) = (y)$ *and* $F\,|\,\alpha = (y)$*, which clearly implies* $F\,|\,\alpha \vdash_1 \mathsf{untouched}_\alpha(x \vee y)$*. Note that the clause* $(\bar{x})$ *is contained in the positive reduct and in the filtered positive reduct because it precludes the assignment* α*.*

If a non-empty assignment α falsifies a formula F, then the filtered positive reduct $\mathsf{f}_\alpha(F)$ is satisfiable. To see this, observe that $\bot \in F\,|\,\alpha$ and therefore $F\,|\,\alpha \vdash_1 \mathsf{untouched}_\alpha(D)$ for every clause $D \in F$. Hence, $\mathsf{f}_\alpha(F) = C$ with C being the clause that precludes α. The ordinary positive reduct does not have this property. The filtered positive reduct identifies exactly the clauses that are set-propagation redundant:

Theorem 34. *Let* F *be a formula,* α *an assignment, and* C *the clause that precludes* α*. Then,* C *is set-propagation redundant with respect to* F *if and only if the filtered positive reduct* $\mathsf{f}_\alpha(F)$ *is satisfiable.*

Proof. For the "only if" direction, suppose C is set-propagation redundant with respect to F, meaning that it contains a non-empty set L of literals such that $F\,|\,\alpha \vdash_1 F\,|\,\alpha_L$. We show that α_L satisfies all clauses of $\mathsf{f}_\alpha(F)$. Let $D' \in \mathsf{f}_\alpha(F)$. By definition, D' is either the clause that precludes α or it is of the form $\mathsf{touched}_\alpha(D)$ for some clause $D \in F$ such that $F\,|\,\alpha \nvdash_1 \mathsf{untouched}_\alpha(D)$. In the former case, D' is clearly satisfied by α_L since α_L must disagree with α. In the latter case, since $F\,|\,\alpha \vdash_1 F\,|\,\alpha_L$, it follows that either $F\,|\,\alpha \vdash_1 D\,|\,\alpha_L$ or α_L satisfies D. Now, it cannot be the case that $F\,|\,\alpha \vdash_1 D\,|\,\alpha_L$ since $var(\alpha_L) = var(\alpha)$ and thus $D\,|\,\alpha_L = \mathsf{untouched}_\alpha(D)$, which would imply $F\,|\,\alpha \vdash_1 \mathsf{untouched}_\alpha(D)$. Therefore, α_L must satisfy D. But then α_L must satisfy $D' = \mathsf{touched}_\alpha(D)$, again since $var(\alpha_L) = var(\alpha)$. It follows that $\mathsf{f}_\alpha(F)$ is satisfiable.

For the "if" direction, assume that α_L satisfies the filtered positive reduct $\mathsf{f}_\alpha(F)$. We show that $F\,|\,\alpha \vdash_1 F\,|\,\alpha_L$. Let $D\,|\,\alpha_L \in F\,|\,\alpha_L$. Since $D\,|\,\alpha_L$ is contained in $F\,|\,\alpha_L$, we know that α_L does not satisfy D and so it does not satisfy $\mathsf{touched}_\alpha(D)$. Hence, $\mathsf{touched}_\alpha(D)$ cannot be contained in $\mathsf{f}_\alpha(F)$, implying that $F\,|\,\alpha \vdash_1 \mathsf{untouched}_\alpha(D)$. But, $D\,|\,\alpha_L = \mathsf{untouched}_\alpha(D)$ since $var(\alpha_L) = var(\alpha)$ and thus it follows that $F\,|\,\alpha \vdash_1 D\,|\,\alpha_L$. We conclude that C is set-propagation redundant with respect to F. $\square$

Since propagation-redundant clauses generalize set-propagation-redundant clauses, it is natural to search for an encoding that characterizes the propagation-redundant clauses. Such an encoding could possibly lead to an even more aggressive pruning of the search space. Finding such an encoding is still part of our future work. However, as we will see in the following, an encoding that characterizes the propagation-redundant clauses must necessarily be large because it has to reason over all possible clauses of a formula.

The positive reduct and the filtered positive reduct yield small formulas that can be easily solved in practice. The downside, however, is that nothing can be learned from their unsatisfiability. This is different for a pruning predicate that encodes propagation redundancy:

Theorem 35. *If a clause $(l_1 \vee \cdots \vee l_k)$ is not propagation redundant with respect to a formula F, then F implies $(\bar{l}_1) \wedge \cdots \wedge (\bar{l}_k)$.*

Proof. Assume $(l_1 \vee \cdots \vee l_k)$ is not propagation redundant with respect to F, or equivalently that all assignments ω with $F|\bar{l}_1 \ldots \bar{l}_k \vdash_1 F|\omega$ agree with $\bar{l}_1 \ldots \bar{l}_k$. Then, no assignment that disagrees with $\bar{l}_1 \ldots \bar{l}_k$ can satisfy F. As a consequence, F implies $(\bar{l}_1) \wedge \cdots \wedge (\bar{l}_k)$. □

By solving a pruning predicate for propagation-redundant clauses, we thus not only detect if the current assignment can be pruned (in case the predicate is satisfiable) but also if the formula is unsatisfiable under any extension of the assignment (in case the predicate is unsatisfiable). We are thus afraid that such an encoding is generally hard to solve and that it might therefore not be useful for SDCL solving in practice.

4.4 Shortening Learned Clauses

If the pruning predicate for a formula F and an assignment α is satisfiable, we know that we can learn the clause that precludes α because it is redundant. In case of the positive reduct and the filtered positive reduct, it is even a propagation-redundant clause because set-blocked clauses and set-propagation-redundant clauses are propagation redundant. To prune the search space even more effectively than by just adding the clause that precludes α, we can in many cases learn a subclause of this clause. The advantage of this is that shorter clauses prune the search space more effectively since they preclude more assignments:

Suppose an SDCL solver is trying to solve a formula F. If α is the current assignment of the solver, it consists of two parts—a part α_{d} of variable assignments that were decisions by the solver and a part α_{u} of assignments that were derived from these decisions via unit propagation on F. If the positive reduct $\mathsf{p}_\alpha(F)$ or the filtered positive reduct $\mathsf{f}_\alpha(F)$ is satisfiable, then we know that the clause that precludes α is propagation redundant with respect to F. Therefore, there exists an assignment ω such that $F|\alpha \vdash_1 F|\omega$. But then, since unit propagation derives all the assignments of α_{u} from $F|\alpha_{\mathrm{d}}$, it must also hold that $F|\alpha_{\mathrm{d}} \vdash_1 F|\omega$, and so the clause that precludes α_{d} is propagation redundant with respect to F. We conclude:

Theorem 36. *Let C be a clause that is propagation redundant with respect to a formula F and let $\alpha = \alpha_{\mathrm{d}} \cup \alpha_{\mathrm{u}}$ be the assignment precluded by C. Assume furthermore that the assignments in α_{u} are derived via unit propagation on $F|\alpha_{\mathrm{d}}$. Then, the clause that precludes α_{d} is propagation redundant with respect to F.*

We can thus learn the clause that precludes only the decision literals in α, and we still end up with a proof in the PR proof system. If we only wanted to preclude α_{d}, we could also just immediately compute the pruning predicate for F and α_{d} instead of the pruning predicate for F and $\alpha_{\mathrm{d}} \cup \alpha_{\mathrm{u}}$. The disadvantage of this, however, is that it makes the pruning predicate less satisfiable, as the following example shows:

Example 24. *Consider the formula* $F = (\bar{x} \vee y) \wedge (x \vee \bar{y})$ *and the assignments* $\alpha = x\,y$, $\alpha_{\mathrm{d}} = x$, *and* $\alpha_{\mathrm{u}} = y$. *Clearly, the unit clause* y *is derived from* $F\,|\,\alpha_{\mathrm{d}}$ *and thus* $\alpha = \alpha_{\mathrm{d}} \cup \alpha_{\mathrm{u}}$. *Now, observe that the positive reduct* $\mathsf{p}_\alpha(F) = F \wedge (\bar{x} \vee \bar{y})$ *is satisfiable, implying that also the filtered positive reduct* $\mathsf{f}_\alpha(F)$ *is satisfiable. On the other hand, the filtered positive reduct* $\mathsf{f}_{\alpha_{\mathrm{d}}}(F) = (x) \wedge (\bar{x})$ *is unsatisfiable, implying that also the positive reduct* $\mathsf{p}_{\alpha_{\mathrm{d}}}(F)$ *is unsatisfiable.*

It thus makes sense to first compute the filtered positive reduct with respect to α and then—in case it is satisfiable—remove the propagated literals to obtain the shorter clause that precludes α_{d}.

4.5 Empirical Evaluation

In the following, we demonstrate that an SDCL solver can prove the unsatisfiability of pigeon hole formulas, Tseitin formulas over expander graphs [Tse68, CS00], and mutilated chessboard problems [McC64, Ale04, DR01]. All three formula families are well-known for not admitting resolution proofs of polynomial size.

Armin Biere implemented an SDCL solver, called SaDiCaL , that can learn propagation-redundant clauses using either the positive reduct or the filtered positive reduct (the source code of SaDiCaL is available at `http://fmv.jku.at/sadical`). The implementation provides a simple but efficient framework to evaluate new SDCL-inspired ideas and heuristics. It closely follows the pseudo-code shown in Figure 4.3 and computes the pruning predicates before making variable assignments via the decision heuristics. This is costly in general, but helps the solver detect redundant clauses as early as possible. Our goal is to determine if short PR proofs can be found automatically.

Two aspects of SDCL are crucial for its performance: the pruning predicates and the decision heuristics. For the pruning predicates, we ran experiments with both the positive reduct and the filtered positive reduct. For the decision heuristics, we chose a heuristic that is different from the *VSIDS* (*variable state independent decaying sum*) [MMZ+01] heuristic, which is the most popular heuristic for CDCL solvers. The idea behind the VSIDS heuristic is to select the variable that occurs most frequently in recent conflict clauses.

Our heuristic generally picks the variable that occurs most frequently in short clauses. Also, it tries to assign only literals that occur in clauses that are touched but not satisfied by the current assignment. There is one more restriction: whenever a (filtered) positive reduct is satisfiable, the heuristic makes all literals in the witness (i.e., in the satisfying assignment of the pruning predicate) that disagree with the current assignment more important than all other literals in the formula. This restriction is removed when the solver backtracks to the first variable (i.e., when a unit clause is learned) and added again when a new propagation-redundant clause is found. We added this restriction because we observed that literals in the witness that disagree with the current assignment typically occur in short propagation-redundant clauses. Making these literals more important than other literals increases the likelihood of learning short clauses.

We compare the solver SaDiCaL in three settings, all with proof logging:

(1) plain CDCL,
(2) SDCL with the positive reduct $\mathsf{p}_\alpha(F)$, and
(3) SDCL with the filtered positive reduct $\mathsf{f}_\alpha(F)$.

Additionally, we include the winner of the 2018 SAT Competition, the CDCL-based solver `MapleLCMDistChronoBT` (short `MLBT`) [NR18].

We first present results regarding Tseitin formulas. In short, Tseitin formulas represent the following graph problem: Given a graph with 0/1-labels for each vertex such that an odd number of vertices has label 1, does there exist a set of edges such that (after removing edges not in the set) every vertex with label 0 has an even degree and every vertex with label 1 has an odd degree? The answer is *no* as the sum of all degrees is always even (the sum of all degrees is twice the number of edges). The resulting formula is therefore unsatisfiable by construction. Tseitin formulas defined over expander graphs are known to require resolution proofs of exponential size. Specialized reasoning, in particular the detection of XOR clauses combined with Gaussian elimination, is known to solve the formulas.

We performed our experiments on a machine using a Xeon E5-2690 CPU with 2.6 GHz and 64 GB memory. The correctness of all the produced proofs was verified with a toolchain involving a formally verified proof checker as presented in Section 3.3. Table 4.1 shows the solver performance on small (Urquhart-s3*), medium (Urquhart-s4*), and large (Urquhart-s5*) Tseitin formulas. Only SaDiCaL with the filtered positive reduct is able to efficiently prove unsatisfiability of all these instances. To the best of our knowledge, SaDiCaL is the first solver that produces machine-checkable proofs of these formulas. Notice that with the ordinary positive reduct it is impossible to solve any of the formulas.

Table 4.2 shows a runtime comparison for the pigeon hole formulas, again including PR proof logging. Although the pigeon hole formulas are hard for resolution, they can be solved efficiently with SDCL using the positive reduct. Notice that the computational costs of the solver with the *filtered* positive reduct are about three to four times as large compared to the solver with the positive reduct. This is caused by the overhead of computing the filtering. The sizes of the PR proofs produced by both versions are similar.

Finally, we performed experiments with the recently released 2018 SAT Competition benchmarks. We expected slow performance on most benchmarks due to the high overhead of solving pruning predicates before making decisions. However, SaDiCaL outperformed the participating solvers on mutilated chessboard problems [McC64] (Table 4.3), which were contributed by Alexey Porkhunov.

For example, with the filtered positive reduct SaDiCaL can prove unsatisfiability of the 18×18 mutilated chessboard in 89 seconds. In the 2018 SAT Competition, all

Table 4.1: Runtime comparison (in seconds) on Tseitin formulas. The columns present the solving times for the solver MLBT as well as for SaDiCaL in CDCL mode (Plain), SDCL with the positive reduct $\mathsf{p}_\alpha(F)$, and SDCL with the filtered positive reduct $\mathsf{f}_\alpha(F)$.

Formula	MLBT	Plain	$\mathsf{p}_\alpha(F)$	$\mathsf{f}_\alpha(F)$
Urquhart-s3-b1	2.95	16.31	> 3600	0.02
Urquhart-s3-b2	1.36	2.82	> 3600	0.03
Urquhart-s3-b3	2.28	2.08	> 3600	0.03
Urquhart-s3-b4	10.74	7.65	> 3600	0.03
Urquhart-s4-b1	86.11	> 3600	> 3600	0.32
Urquhart-s4-b2	154.35	183.77	> 3600	0.11
Urquhart-s4-b3	258.46	129.27	> 3600	0.16
Urquhart-s4-b4	> 3600	> 3600	> 3600	0.14
Urquhart-s5-b1	> 3600	> 3600	> 3600	1.27
Urquhart-s5-b2	> 3600	> 3600	> 3600	0.58
Urquhart-s5-b3	> 3600	> 3600	> 3600	1.67
Urquhart-s5-b4	> 3600	> 3600	> 3600	2.91

Table 4.2: Runtime comparison (in seconds) on pigeon hole formulas.

Formula	MLBT	Plain	$\mathsf{p}_\alpha(F)$	$\mathsf{f}_\alpha(F)$
hole20	> 3600	> 3600	0.26	0.49
hole30	> 3600	> 3600	1.96	4.03
hole40	> 3600	> 3600	9.02	19.54
hole50	> 3600	> 3600	28.63	65.90

Table 4.3: Runtime comparison (in seconds) on mutilated chessboard problems.

Formula	MLBT	Plain	$\mathsf{p}_\alpha(F)$	$\mathsf{f}_\alpha(F)$
mchess_15	51.53	2480.67	> 3600	13.14
mchess_16	380.45	2115.75	> 3600	15.52
mchess_17	2418.35	> 3600	> 3600	25.54
mchess_18	> 3600	> 3600	> 3600	43.88

other solvers—apart from CaDiCaL (a CDCL solver by Armin Biere) solving it in 828 seconds—timed out after 5000 seconds.

Considering the outcome of our experiments, we believe that SDCL—when combined with sophisticated heuristics and encodings—is a promising SAT-solving paradigm for formulas that are too hard for ordinary CDCL solvers. Moreover, proofs of challenging problems can be enormous in size, such as the 2 petabytes proof of Schur Number Five [Heu18]. SDCL improvements have the potential to produce substantially smaller proofs.

CHAPTER 5

Redundant Clauses in First-Order Logic

After having considered propositional logic, we now move on to the more expressive first-order logic case. The theme of our work, however, stays the same: we still focus on clause redundancy.

As we have seen in previous chapters, research on SAT solving has given rise to a wide variety of redundancy properties that play an important role in state-of-the-art reasoning engines. For many of these redundancy properties, however, it was unclear whether or not they could be lifted to the level of first-order logic. We address this issue and introduce the principle of *implication modulo resolution*—a first-order generalization of *quantified implied outer resolvents* as introduced by Heule et al. [HSB16] in the context of quantified Boolean formulas. The principle of implication modulo resolution allows us to lift several redundancy properties in a uniform way.

Informally, a clause C is *implied modulo resolution* by a CNF formula F if C contains a literal such that all resolvents upon this literal are implied by F. Here, by *all resolvents* we mean all first-order resolvents with clauses in F. In other words, although F might not necessarily imply the clause C itself, it implies all the conclusions that can be derived with C via resolution upon one of its literals. We show that this suffices to ensure that C is redundant with respect to F in first-order logic *without* equality.

Using implication modulo resolution, we lift various redundancy properties to first-order logic without equality. These redundancy properties include *blocked clauses* (BC) [Kul99], *covered clauses* (CC) [HJB10b], *asymmetric tautologies* (AT) [HJB10a], *resolution asymmetric tautologies* (RAT) [JHB12], and *resolution-subsumed clauses* (RS) [JHB12]. None of these redundancy properties have been available in first-order logic before.

Although in previous chapters we focused mainly on the *addition* of redundant clauses, the *elimination* of redundant clauses can also significantly improve the performance of modern reasoning engines [HJL$^+$15]. We therefore consider clause-elimination techniques based on the lifted redundancy properties and analyze if they are confluent. Intuitively, confluence of a technique tells us that the order in which we eliminate clauses from a formula is not relevant to the final outcome of the elimination procedure.

After this, we present the principle of *implication modulo flat resolution*, a variant of implication modulo resolution for first-order logic *with* equality. We show how the use of implication modulo flat resolution yields a short soundness proof for the existing preprocessing technique of *predicate elimination* [KK16]. Moreover, we use implication modulo flat resolution to derive a variant of blocked clauses—called *equality-blocked clauses*—that guarantees redundancy even in first-order logic with equality.

Finally, we present an application of blocked clauses and equality-blocked clauses in first-order logic: a preprocessing tool that eliminates (equality) blocked clauses from formulas to speed up first-order theorem provers. We present an empirical evaluation showing that blocked-clause elimination is a beneficial preprocessing technique that can significantly boost performance. Blocked-clause elimination is now part of the theorem prover Vampire [KV13], which has won the FOF (First-Order Form theorems) division of the CASC competition [SU16] for automated theorem proving each year since 2002.

5.1 First-Order Logic Without Equality

We assume the reader to be familiar with the basics of first-order logic. As usual, formulas of a first-order language $\mathcal{L}$ are built using predicate symbols, function symbols, and constants from some given denumerable alphabet together with logical connectives, quantifiers, and variables. We use the letters $P, Q, R, S, \dots$ as predicate symbols and the letters $f, g, h, \dots$ as non-constant function symbols. Moreover, we use the letters $a, b, c, \dots$ for constants and the letters $x, y, z, u, v, \dots$ for variables (possibly with subscripts).

As in propositional logic, we consider formulas in conjunctive normal form, which are defined as follows. An *atom* is an expression $P(t_1, \dots, t_n)$ where P is a predicate symbol of arity n and $t_1, \dots t_n$ are terms built from constants, variables, and function symbols as usual. Literals, clauses, and formulas are then defined analogously to propositional logic, allowing atoms instead of only propositional variables: A *literal* is either an atom (a *positive literal*) or the negation $\bar{A}$ of an atom A (a *negative literal*). A disjunction of literals is a *clause*. A conjunction of clauses is a *formula*. An expression (i.e., a term, literal, formula, etc.) is *ground* if it contains no variables. For a literal L, we define its complement $\bar{L}$ as $\bar{A}$ if $L = A$ and as $\bar{L} = A$ if $L = \bar{A}$, where A is an atom. Without loss of generality, clauses are assumed to be variable disjoint. Variables occurring in a CNF formula are implicitly universally quantified. We treat CNF formulas as sets of clauses and clauses as multisets of literals. A clause is a *tautology* if it contains both L and $\bar{L}$ for some literal L.

Regarding the semantics of first-order logic, we use the standard notions of *interpretation*, *model*, *validity*, *satisfiability*, and *logical equivalence*. As in propositional logic, we say that two formulas are *equisatisfiable* if they are either both satisfiable or both unsatisfiable. A *propositional assignment* is a mapping from ground atoms to the truth values 1 (*true*) and 0 (*false*). Accordingly, a set of ground clauses is *propositionally satisfiable* if there exists a propositional assignment that satisfies F under the propositional semantics, treating ground atoms like propositional variables.

A *substitution* is a mapping from variables to terms that agrees with the identity function on all but finitely many variables. Let σ be a substitution. The domain $dom(\sigma)$ of σ is the set of variables for which $\sigma(x) \neq x$. The range $ran(\sigma)$ of σ is the set $\{\sigma(x) \mid x \in dom(\sigma)\}$. A substitution is *ground* if its range consists only of ground terms. Every substitution σ can be extended to a mapping $\hat{\sigma}$ over terms by defining $\hat{\sigma}(x) = \sigma(x)$ for variables x, and $\hat{\sigma}(f(t_1, \ldots, t_n)) = f(\hat{\sigma}(t_1), \ldots, \hat{\sigma}(t_n))$ for non-variable terms $f(t_1, \ldots, t_n)$. As common, given an expression E, we write $E\sigma$ for $\hat{\sigma}(E)$. For instance, if $\sigma = \{x \mapsto g(a)\}$, then $L(x, f(x))\sigma = L(g(a), f(g(a)))$. If $E\sigma$ is ground, it is a *ground instance* of E. The composition $\sigma\tau$ of two substitutions is defined as $x\sigma\tau = \hat{\tau}(\sigma(x))$ for all variables x.

A substitution σ is a *unifier* of the expressions $E_1, \ldots, E_n$ if $E_1\sigma = \cdots = E_n\sigma$. For substitutions σ and τ, we say that σ is *more general* than τ if there exists a substitution λ such that $\sigma\lambda = \tau$. Furthermore, σ is a *most general unifier* (*mgu*) of $E_1, \ldots, E_n$ if, for every unifier τ of $E_1, \ldots, E_n$, σ is more general than τ. It is well-known that whenever a set of expressions is unifiable, there exists an idempotent most general unifier of this set.

We make use of a popular variant of Herbrand's Theorem [Fit96]:

Theorem 37. *A formula F is satisfiable if and only if every finite set of ground instances of clauses in F is propositionally satisfiable.*

Our notion of clause redundancy is analogous to the one we used in propositional logic:

Definition 34. *A clause C is* redundant *with respect to a formula F if F and $F \land C$ are equisatisfiable.*

The first-order notion of a *resolvent* involves most general unifiers:

Definition 35. *Given two clauses $C = L_1 \lor \cdots \lor L_k \lor C'$ and $D = N_1 \lor \cdots \lor N_l \lor D'$ such that the literals $L_1, \ldots, L_k, \bar{N}_1, \ldots, \bar{N}_l$ are unifiable by an mgu σ, the clause $C'\sigma \lor D'\sigma$ is a* resolvent *of C and D. If $k = l = 1$, it is a* binary resolvent *of C and D* upon *L_1.*

Unlike in propositional logic, there can exist multiple resolvents of two clauses upon a single literal:

Example 25. *Consider the clauses $P(x) \lor P(y) \lor R(x, y)$ and $\bar{P}(a) \lor Q(a)$. The clauses $P(y) \lor R(a, y) \lor Q(a)$, $P(x) \lor R(x, a) \lor Q(a)$, and $R(a, a) \lor Q(a)$ are resolvents. The first two resolvents are binary resolvents whereas the third one is not.*

5.1.1 Implication Modulo Resolution

We can now proceed to define the principle of implication modulo resolution. The definition of implication modulo resolution relies on the notion of an *L-resolvent*. Intuitively, an L-resolvent is obtained by resolving only upon a single literal of the left-hand clause:

Definition 36. *Given two clauses $C = L \lor C'$ and $D = N_1 \lor \cdots \lor N_l \lor D'$ such that the literals $L, \bar{N}_1, \ldots, \bar{N}_l$ are unifiable by an mgu σ, the clause $C'\sigma \lor D'\sigma$ is called* L-resolvent *of C and D.*

Example 26. *Let $C = P(x) \lor P(a) \lor Q(x,a)$, $D = \bar{P}(y) \lor \bar{P}(z) \lor R(y,z)$, and let $L = P(x)$. Then, the substitution $\{y \mapsto x, z \mapsto x\}$ is an mgu of $P(x)$, $P(y)$, and $P(z)$. Therefore, $P(a) \lor Q(x,a) \lor R(x,x)$ is an L-resolvent of C and D. Also the resolvent $P(a) \lor Q(x,a) \lor \bar{P}(z) \lor R(x,z)$, obtained by using the mgu $\{y \mapsto x\}$ of $P(x)$ and $P(y)$ is an L-resolvent of C and D. However, the resolvent $Q(a,a) \lor R(a,a)$, obtained by using the mgu $\{x \mapsto a, y \mapsto a, z \mapsto a\}$ of $P(x)$, $P(a)$, $P(y)$, and $P(z)$ is not an L-resolvent as it resolves away the literal $P(a)$ from the left-hand clause C.*

Before we next define the principle of implication modulo resolution, we want to highlight that whenever we say that a formula F *implies* a clause C, we mean that every model of F is a model of C, that is, $F \models C$.

Definition 37. *A clause C is* implied modulo resolution *by a formula F if C contains a literal L such that all L-resolvents of C, with clauses in F, are implied by F.*

We say that C is implied modulo resolution *upon L* by F. A simple example for clauses that are implied modulo resolution are clauses with *pure literals*. A pure literal is a literal whose predicate symbol occurs in only one polarity in the whole formula. Since there are no resolvents upon such a literal, the containing clause is trivially implied modulo resolution. The following example is a little more involved:

Example 27. *Let $C = P(x) \lor Q(x)$ and*

$$F = \{\bar{P}(y) \lor R(y),\ R(z) \lor S(z),\ \bar{S}(u) \lor Q(u)\}.$$

There is one $P(x)$-resolvent of C, namely $Q(x) \lor R(x)$, obtained by resolving C with $\bar{P}(y) \lor R(y)$. Clearly, this resolvent is implied by the clauses $R(z) \lor S(z)$ and $\bar{S}(u) \lor Q(u)$. Therefore, F implies C modulo resolution upon $P(x)$.

In the following, we prove that implication modulo resolution ensures redundancy, i.e., if a clause C is implied modulo resolution by a formula F, then C is redundant with respect to F. In the proof, we use Herbrand's Theorem (Theorem 37), which tells us that a formula F is satisfiable if and only if all finite sets of ground instances of clauses in F are propositionally satisfiable.

To prove that the satisfiability of F implies the satisfiability of $F \land C$, we proceed as follows: Given a finite set of ground instances of clauses in $F \land C$, we can obtain a satisfying propositional assignment of this set from an assignment that satisfies all the ground instances of clauses in F. The latter assignment is guaranteed to exist because F is satisfiable. The key idea behind the modification of this assignment is to flip the truth values of certain (ground) literals, just as we did in propositional logic in previous chapters. We illustrate this on the following example:

Example 28. *Consider again C and F from Example 27 and let $C' = P(a) \vee Q(a)$ be a ground instance of C. Let furthermore $F' = \{\bar{P}(a) \vee R(a),\ R(a) \vee S(a),\ \bar{S}(a) \vee Q(a)\}$ be a finite set of ground instances of F (in fact, F' is even a ground instance of F). Clearly, F' is propositionally satisfied by the assignment $\alpha = \bar{P}(a)R(a)\bar{S}(a)\bar{Q}(a)$, but α falsifies C'. However, we can turn α into a satisfying assignment of C' by flipping the truth value of $P(a)$—the instance of the literal upon which C is implied modulo resolution. The resulting assignment $\alpha' = P(a)R(a)\bar{S}(a)\bar{Q}(a)$ could possibly falsify the clause $\bar{P}(a) \vee R(a)$ since it contains $\bar{P}(a)$, which is not satisfied anymore. But, $\bar{P}(a) \vee R(a)$ stays true since $R(a)$ is satisfied by α'. Therefore, α' satisfies $F' \wedge C'$.*

In the above example, it is not a coincidence that $\bar{P}(a) \vee R(a)$ is still satisfied after flipping the truth value of $P(a)$. The intuitive explanation is as follows: The clause $Q(a) \vee R(a)$ is a ground instance of the $P(x)$-resolvent $Q(x) \vee R(x)$ of C and $\bar{P}(y) \vee R(y)$, and we know that this resolvent is implied by F. Therefore, since α satisfies all the ground instances of clauses in F, it should also satisfy $Q(a) \vee R(a)$. But, since α does not satisfy $Q(a)$ (because α falsifies $C' = P(a) \vee Q(a)$), it must satisfy $R(a)$, and so it satisfies $\bar{P}(a) \vee R(a)$. Finally, since α' disagrees with α only on $P(a)$, it also satisfies $R(a)$. The following lemma formalizes this observation:

Lemma 38. *Let C be a clause that is implied modulo resolution upon L by F, and let α be an assignment that propositionally satisfies all ground instances of clauses in F but falsifies a ground instance $C\lambda$ of C. Then, the assignment α', obtained from α by flipping the truth value of $L\lambda$, still satisfies all ground instances of clauses in F.*

Proof. Let $D\tau$ be a ground instance of a clause $D \in F$ and suppose α satisfies $D\tau$. If $D\tau$ does not contain $\bar{L}\lambda$, it is trivially satisfied by α'. Assume therefore that $\bar{L}\lambda \in D\tau$ and let $N_1, \dots, N_l$ be all the literals in D such that $N_i\tau = \bar{L}\lambda$ for $1 \leq i \leq l$. Then, the substitution $\lambda\tau = \lambda \cup \tau$ (note that C and D are variable disjoint by assumption) is a unifier of $L, \bar{N}_1, \dots, \bar{N}_l$. Hence, $R = (C \setminus \{L\})\sigma \vee (D \setminus \{N_1, \dots, N_l\})\sigma$, with σ being an *mgu* of $L, \bar{N}_1, \dots, \bar{N}_l$, is an L-resolvent of C and thus implied by F.

As σ is most general, there exists a substitution γ such that $\sigma\gamma = \lambda\tau$. Therefore,

$$\begin{array}{rlcl} & (C \setminus \{L\})\sigma\gamma & \vee & (D \setminus \{N_1, \dots, N_l\})\sigma\gamma \\ = & (C \setminus \{L\})\lambda\tau & \vee & (D \setminus \{N_1, \dots, N_l\})\lambda\tau \\ = & (C \setminus \{L\})\lambda & \vee & (D \setminus \{N_1, \dots, N_l\})\tau \end{array}$$

is a ground instance of R and so it must be satisfied by α. Thus, since α falsifies $C\lambda$, it must satisfy a literal $L'\tau \in (D \setminus \{N_1, \dots, N_l\})\tau$. But, as all the literals in

$(D \setminus \{N_1, \dots, N_l\})\tau$ are different from $\bar{L}\lambda$, flipping the truth value of $L\lambda$ does not affect the truth value of $L'\tau$. It follows that α' satisfies $L'\tau$ and thus it satisfies $D\tau$. □

We can therefore satisfy a previously falsified ground instance $C\lambda$ of C without falsifying ground instances of clauses in F, by flipping the truth value of $L\lambda$—the ground instance of the literal L upon which C is implied modulo resolution. Still, as the following example shows, there could be other ground instances of C that contain the complement $\bar{L}\lambda$ of $L\lambda$. These ground instances can potentially be falsified when making $L\lambda$ true:

Example 29. *Suppose a formula F implies a clause $C = \bar{P}(x) \vee P(f(x))$ modulo resolution upon $P(f(x))$ and consider the two ground instances $C_1 = \bar{P}(a) \vee P(f(a))$ and $C_2 = \bar{P}(f(a)) \vee P(f(f(a)))$ of C. The assignment $P(a)\bar{P}(f(a))\bar{P}(f(f(a)))$ falsifies C_1, but we can satisfy C_1 by flipping the truth value of $P(f(a))$—which is the ground instance of $P(f(x))$—to obtain the assignment $P(a)P(f(a))\bar{P}(f(f(a)))$. However, by flipping the truth value of $P(f(a))$, we falsified the other ground instance C_2 of C.*

In the proof of Theorem 39 below, we show that this is actually not a serious problem. The key idea is to repeatedly satisfy ground instances of the literal upon which the clause is implied modulo resolution, until we finally obtain a satisfying assignment of all ground instances of the clause. In the above example, for instance, we can continue by flipping the truth value of $P(f(f(a))$ to obtain a satisfying assignment of both C_1 and C_2.

Theorem 39. *If a formula F implies a clause C modulo resolution, then C is redundant with respect to F.*

Proof. Assume that F implies C modulo resolution upon L and that F is satisfiable. We show that $F \wedge C$ is satisfiable. By Herbrand's theorem (Theorem 37), it suffices to show that every finite set of ground instances of clauses in $F \wedge C$ is propositionally satisfiable. Let therefore F' and F_C be finite sets of ground instances of clauses in F and $\{C\}$, respectively. Since F is satisfiable, there exists an assignment α that propositionally satisfies all ground instances of clauses in F and thus it clearly satisfies F'. Assume now that α falsifies some ground instances of C that are contained in F_C.

By Lemma 38, for every falsified ground instance $C\lambda$ of C, we can turn α into a satisfying assignment of $C\lambda$ by flipping the truth value of $L\lambda$, and this flipping does not falsify any ground instances of clauses in F. The only clauses that could possibly be falsified are other ground instances of C that contain the literal $\bar{L}\lambda$. But, once an instance $L\tau$ of L is true in a ground instance $C\tau$ of C, $L\tau$ cannot (later) be falsified by making other instances of L true. As there are only finitely many clauses in F_C, we can therefore turn α into a satisfying assignment of $F' \cup F_C$ by repeatedly making ground instances of C true by flipping the truth values of their instances of L, until all ground instances of C are satisfied. We conclude that all finite sets of ground instances of clauses in $F \wedge C$ are propositionally satisfiable and so $F \wedge C$ is satisfiable. □

For example, the clause C in Example 27 is redundant with respect to F since it is implied modulo resolution by F. In what follows, we lift several redundancy properties from propositional logic to first-order logic. Thereby, Theorem 39 will help us to prove their redundancy. We start with blocked clauses, as both resolution asymmetric tautologies and covered clauses (which we lift later) can be seen as generalizations of blocked clauses.

5.1.2 Blocked Clauses

We have discussed blocked clauses extensively before, in the section on locally redundant clauses in propositional logic. Remember that in propositional logic, a clause C is *blocked* in a formula F if it contains a literal such that all binary resolvents of C upon this literal are tautologies. In first-order logic, we replace the notion of a binary resolvent by that of an L-resolvent:

Definition 38. *A clause C is* blocked *in a formula F if it contains a literal L such that all L-resolvents of C, with clauses in F, are tautologies.*

We say that C is blocked *by L* in F.

Example 30. *Consider the clause $C = P(x) \vee \bar{Q}(x)$ and the formula $F = \{\bar{P}(y) \vee Q(y)\}$. There is only one $P(x)$-resolvent of C, namely the tautology $\bar{Q}(x) \vee Q(x)$, obtained by using the mgu $\sigma = \{y \mapsto x\}$. Therefore, C is blocked by $P(x)$ in F.*

As tautologies are trivially implied by every formula, blocked clauses are implied modulo resolution. The redundancy of blocked clauses in first-order logic is therefore a consequence of Theorem 39:

Theorem 40. *If a clause is blocked in a formula F, it is redundant with respect to F.*

5.1.3 Asymmetric Tautologies and RATs

We have already discussed the propositional notions of asymmetric tautologies and RATs in Section 2.2.1 on page 19. Remember that an asymmetric tautology is a clause that can be turned into a tautology by repeatedly adding *asymmetric literals* to it. In propositional logic, a literal L is an asymmetric literal with respect to a clause C in a formula F if there exists a clause $D \vee \bar{L} \in F$ such that D subsumes C, i.e., $D \subseteq C$. The addition of an asymmetric literal L to a clause C yields a clause that is logically equivalent to C in the sense that $F \models C \leftrightarrow (C \vee L)$ [HJB10a].

In first-order logic, a clause C *subsumes* a clause D if there exists a substitution λ such that $C\lambda \subseteq D$. This motivates the following first-order variants of asymmetric literals and asymmetric tautologies:

Definition 39. *A literal L is an* asymmetric literal *with respect to a clause C in a formula F if there exist a clause $D \vee K \in F$ and a substitution λ such that $D\lambda \subseteq C$ and $L = \bar{K}\lambda$.*

Example 31. *Consider the formula $F = \{P(y) \vee Q(y) \vee \bar{S}(y)\}$ and the clause $C = P(x) \vee Q(x) \vee R(x)$. The literal $S(x)$ is an asymmetric literal with respect to C in F since, for $\lambda = \{y \mapsto x\}$, $(P(y) \vee Q(y))\lambda \subseteq C$ and $S(x) = S(y)\lambda$.*

First-order asymmetric-literal addition is harmless insofar as the original clause C can be obtained from $C \vee L$ and $D \vee K$ via an application of the subsumption-resolution rule, which—as the name suggests—combines subsumption and resolution [BG01]:

$$\frac{C' \vee D\lambda \vee \bar{K}\lambda \qquad D \vee K}{C' \vee D\lambda}$$

Clearly, the consequence of the subsumption-resolution rule is implied by its premises. In case L is an asymmetric literal, we have that $L = \bar{K}\lambda$ and $D\lambda \subseteq C$, hence C is of the form $C' \vee D\lambda \vee \bar{K}\lambda$ and thus we can derive $C = C' \vee D\lambda$ from $C \vee L$ and $D \vee K$. We thus get:

Lemma 41. *Let F be a formula, C a clause, and L an asymmetric literal with respect to C in F. Then, $F \models C \leftrightarrow (C \vee L)$.*

As in propositional logic, an asymmetric tautology is a clause that can be turned into a tautology by adding asymmetric literals (*asymmetric-literal addition*, ALA):

Definition 40. *A clause C is an* asymmetric tautology *in a formula F if there exists a sequence $L_1, \dots, L_n$ of literals such that $C \vee L_1 \vee \dots \vee L_n$ is a tautology and each L_i is an asymmetric literal with respect to $C \vee L_1 \vee \dots \vee L_{i-1}$ in F.*

Example 32. *Consider the formula $F = \{R(z) \vee S(z),\ \bar{S}(u) \vee Q(u)\}$ and the clause $C = Q(x) \vee R(x)$. The subclause $R(z)$ of $R(z) \vee S(z)$ subsumes $R(x)$ via $\{z \mapsto x\}$ and so $\bar{S}(x)$ is an asymmetric literal with respect to C. We add it to C and obtain the clause $Q(x) \vee R(x) \vee \bar{S}(x)$. After this, the subclause $\bar{S}(u)$ of $\bar{S}(u) \vee Q(u)$ subsumes $\bar{S}(x)$ via $\{u \mapsto x\}$ and thus $\bar{Q}(x)$ can be added to obtain the tautology $Q(x) \vee R(x) \vee \bar{S}(x) \vee \bar{Q}(x)$. We conclude that C is an asymmetric tautology in F.*

Note that in automatic theorem proving, we prefer short clauses over long ones, since the short clauses are usually stronger. Therefore, when performing asymmetric-tautology elimination, the asymmetric-literal additions are not meant to be permanent: We first add asymmetric literals and then test whether the resulting clause is a tautology. If so, we remove the clause; if not, we undo the asymmetric-literal additions to shrink the clause back to its original size. We next show that asymmetric tautologies are implied:

Theorem 42. *If C is an asymmetric tautology in F, it is implied by F.*

Proof. Suppose C is an asymmetric tautology in F, meaning that there exists a sequence $L_1, \dots, L_n$ of literals such that $C \vee L_1 \vee \dots \vee L_n$ is a tautology and each L_i is an asymmetric literal with respect to the clause $C \vee L_1 \vee \dots \vee L_{i-1}$ in F. By the repeated application of Lemma 41 (an easy induction argument), it follows that $F \models (C \leftrightarrow C \lor L_1 \lor \dots \lor L_n)$. But then, since $C \lor L_1 \lor \dots \lor L_n$ is a tautology, it trivially holds that $F \models C \lor L_1 \lor \dots \lor L_n$ and so $F \models C$. $\square$

Unlike in propositional logic, the first-order variant of asymmetric-literal addition is not guaranteed to terminate. Consider the following example:

Example 33. *Let $C = P(a)$ and $F = \{P(x) \lor \bar{P}(f(x))\}$. Then, since $P(x)$ subsumes $P(a)$ via $\lambda = \{x \mapsto a\}$, we can add the asymmetric literal $P(f(a))$ to obtain $P(a) \lor P(f(a))$. After this, we can add $P(f(f(a))$ via $\lambda = \{x \mapsto f(a)\}$, then $P(f(f(f(a))))$ and so on. This can be repeated infinitely many times.*

A resolution asymmetric tautology in first-order logic is then a clause C that contains a literal L such that all L-resolvents of C are asymmetric tautologies:

Definition 41. *A clause C is a* resolution asymmetric tautology (RAT) *in a formula F if it contains a literal L such that all L-resolvents of C, with clauses in F, are asymmetric tautologies in F.*

We say that C is a RAT *on* L in F.

Example 34. *Consider the clause $C = P(x) \lor Q(x)$ and the following formula $F = \{\bar{P}(y) \lor R(y),\ R(z) \lor S(z),\ \bar{S}(u) \lor Q(u)\}$ (cf. Example 27). There is one $P(x)$-resolvent of C, namely $Q(x) \lor R(x)$, obtained by resolving with $\bar{P}(y) \lor R(y)$. The formula F is a superset of the formula from Example 32 in which $Q(x) \lor R(x)$ is an asymmetric tautology. Thus, $Q(x) \lor R(x)$ is also an asymmetric tautology here: The subclause $R(z)$ of the clause $R(z) \lor S(z)$ subsumes $R(x)$ via $\{z \mapsto x\}$ and so $\bar{S}(x)$ is an asymmetric literal with respect to $Q(x) \lor R(x)$. We add it to C and obtain the clause $Q(x) \lor R(x) \lor \bar{S}(x)$. After this, the subclause $\bar{S}(u)$ of $\bar{S}(u) \lor Q(u)$ subsumes $\bar{S}(x)$ via $\{u \mapsto x\}$ and so $\bar{Q}(x)$ can be added to obtain the tautology $Q(x) \lor R(x) \lor \bar{S}(x) \lor \bar{Q}(x)$. It follows that C is a* RAT *in F.*

Theorem 43. *If a clause C is a* RAT *in a formula F, it is redundant with respect to F.*

Proof. Assume that C is a RAT in F. Then, every L-resolvent of C with clauses in F is an asymmetric tautology in F and therefore, by Theorem 42, implied by F. It follows that C is implied modulo resolution upon L by F and thus, by Theorem 39, C is redundant with respect to F. $\square$

5.1.4 Covered Clauses

In contrast to blocked clauses and resolution asymmetric tautologies, we haven't discussed covered clauses in previous chapters. We thus first discuss the notions of covered literals and covered clauses from propositional logic and then lift them to the first-order level. Informally, a clause C is *covered* in a propositional formula F if the addition of so-called *covered literals* turns it into a blocked clause. A clause C *covers* a literal K in F if C

contains a literal L such that all non-tautological resolvents of C upon L contain K. The crucial property of covered literals is that they can be added to C without affecting satisfiability [HJB10b]. More precisely, given a formula F, a clause C, and a literal K that is covered by C in F, it holds that $F \wedge C$ and $F \wedge (C \vee K)$ are equisatisfiable.

Example 35. *Consider the clause $C = P$ and the formula $F = \{\bar{P} \vee \bar{Q} \vee R,\ \bar{P} \vee \bar{Q} \vee S\}$. There are two resolvents of C upon P, namely $\bar{Q} \vee R$ and $\bar{Q} \vee S$. As $\bar{Q}$ is contained in both resolvents, it is covered by C in F. Therefore, $F \wedge P$ and $F \wedge (P \vee \bar{Q})$ are equisatisfiable.*

We next introduce a first-order variant of covered literals. It is based on the notion of a *non-recursive* literal:

Definition 42. *A literal L is* recursive *in a clause C if C contains a literal K such that K and L have the same predicate symbol but opposite polarity.*

If C is clear from the context, we just say that L is recursive. If L is not recursive in C, we say that it is *non-recursive*.

Example 36. *The literal $\bar{P}(x)$ is recursive in the clause $\bar{P}(x) \vee P(f(x)) \vee Q(x)$; it is non-recursive in the clause $\bar{P}(x) \vee \bar{P}(f(x)) \vee Q(x)$.*

Using the notion of a non-recursive literal, we can now define covered literals:

Definition 43. *A clause C* covers *a literal K in a formula F if C contains a non-recursive literal L such that all non-tautological L-resolvents of C, with clauses in F, contain K.*

Note that our definition of covered literals implies that all the non-tautological L-resolvents of C must contain *exactly* K, meaning that even the variable names occurring in K have to be identical across all L-resolvents. Although there might be other generalizations of covered literals which are not as restrictive, we adopt this definition due to its simplicity.

Example 37. *Consider the clause $C = P(f(x))$ and the formula*

$$F = \{\bar{P}(y) \vee Q(y) \vee R(y),\ \bar{P}(z) \vee Q(z) \vee S(z)\}.$$

There are two $P(f(x))$-resolvents of C: $Q(f(x)) \vee R(f(x))$, obtained by using the mgu $\{y \mapsto f(x)\}$, and $Q(f(x)) \vee S(f(x))$, obtained by using the mgu $\{z \mapsto f(x)\}$. Since $Q(f(x))$ is contained in both resolvents, it is covered by C in F.

As we will show below, the addition of a covered literal to the clause that covers it has no effect on satisfiability in the sense that $F \wedge C$ and $F \wedge (C \vee K)$ are equisatisfiable. The following example shows that this would not be the case if we did not require L to be non-recursive:

Example 38. *Consider the clause $C = \bar{P}(x) \vee P(f(x))$ and the formula*

$$F = \{\bar{P}(y) \vee Q(y),\ P(a),\ \bar{Q}(f(f(a))\}.$$

The literal $Q(f(x))$ is contained in the (only) $P(f(x))$-resolvent $\bar{P}(x) \vee Q(f(x))$ of C with clauses in F. However, $F \wedge C$ is unsatisfiable whereas $F \wedge (C \vee Q(f(x)))$ is satisfiable.

Lemma 44. *If a clause C covers a literal K in a formula F, then $F \wedge C$ and $F \wedge (C \vee K)$ are equisatisfiable.*

Proof. Assume that C covers K in F, meaning that C contains a non-recursive literal L such that K is contained in all non-tautological L-resolvents of C with clauses in F. First, we add $(C\tau \vee K\tau)$ to $F \wedge C$, with τ being a renaming that replaces the variables in $(C \vee K)$ by fresh variables not occurring in F. Since $(C\tau \vee K\tau)$ is subsumed by C, the formulas $F \wedge C$ and $F \wedge C \wedge (C\tau \vee K\tau)$ are equisatisfiable. We next show that C is redundant with respect to $F \wedge (C\tau \vee K\tau)$ and that it can therefore by removed without affecting the satisfiability status. To do so, we show that $F \wedge (C\tau \vee K\tau)$ implies C modulo resolution upon L. As $F \wedge (C\tau \vee K\tau)$ and $F \wedge (C \vee K)$ are clearly equivalent, the claim then follows.

We have to show that all L-resolvents of C with clauses in F are implied by $F \wedge (C\tau \vee K\tau)$ (note that we do not need to consider L-resolvents with $(C\tau \vee K\tau)$ since the non-recursiveness of L implies that such resolvents do not exist). Since tautological L-resolvents are trivially implied, we consider only non-tautological ones. Let $C'\sigma \vee D'\sigma$ be a non-tautological L-resolvent of $C = C' \vee L$ with a clause $D = D' \vee N_1 \vee \cdots \vee N_k \in F$, where σ is an (idempotent) *mgu* of the literals $L, \bar{N}_1, \ldots, \bar{N}_k$. Since K is covered by C in F, the resolvent $C'\sigma \vee D'\sigma$ contains K, and K is of the form $P\sigma$ for some literal $P \in C' \vee D'$.

To prove that $C'\sigma \vee D'\sigma$ is implied by $F \wedge (C\tau \vee K\tau)$, we show that it can be obtained from clauses in $F \wedge (C\tau \vee K\tau)$ via resolution, substitution, and factoring: Consider the clauses $(C\tau \vee K\tau) = (C'\tau \vee L\tau \vee K\tau)$ and $D = (D' \vee N_1 \vee \cdots \vee N_k)$. Since the literals $L, \bar{N}_1, \ldots, \bar{N}_k$ are unified by σ and since $dom(\tau^{-1}) \cap var(D) = \emptyset$, it follows that $L\tau$ and $\bar{N}_1, \ldots, \bar{N}_k$ are unified by $\tau^{-1}\sigma$ (note that the inverse function τ^{-1} of τ is a valid substitution since $ran(\tau)$ consists only of fresh variables). Therefore, there exists an *mgu* σ' of $L\tau$ and $\bar{N}_1, \ldots, \bar{N}_k$. Hence, the clause $(C'\tau \vee K\tau \vee D')\sigma'$ is an $L\tau$-resolvent of $(C\tau \vee K\tau)$ and D. Now, since σ' is most general, there exists a substitution γ such that $\sigma'\gamma = \tau^{-1}\sigma$. But then,

$$\begin{aligned} &(C'\tau \vee K\tau \vee D')\sigma'\gamma \\ = {}&(C'\tau \vee K\tau \vee D')\tau^{-1}\sigma \\ = {}&(C'\sigma \vee K\sigma \vee D'\sigma), \end{aligned}$$

from which we obtain $(C'\sigma \vee D'\sigma)$ by factoring, since $K \in C'\sigma \vee D'\sigma$ and $K\sigma = P\sigma\sigma = P\sigma = K$ (note that $D'\tau^{-1} = D'$ since $var(D') \cap var(\tau^{-1}) = \emptyset$). We conclude that $F \wedge (C\tau \vee K\tau)$ implies C modulo resolution upon L. $\square$

Similar to asymmetric-literal addition, the addition of covered literals in first-order logic is also not guaranteed to terminate. Consider the following example:

Example 39. *Let $C = P(a)$ and $F = \{\bar{P}(x) \vee P(f(x))\}$. Then, there exists one $P(a)$-resolvent of C, namely $P(f(a))$. Therefore, $P(f(a))$ is covered by C and thus it can be added to C to obtain $P(a) \vee P(f(a))$. Now, there is one $P(f(a))$-resolvent of $P(a) \vee P(f(a))$, namely $P(a) \vee P(f(f(a)))$, and thus $P(f(f(a)))$ can be added. This addition of covered literals can be repeated infinitely often.*

A clause C is then covered in a formula F if the repeated addition of covered literals can turn it into a blocked clause:

Definition 44. *A clause C is* covered *in a formula F if there exists a sequence $K_1, \dots, K_n$ of literals such that each K_i is covered by $C \vee K_1 \vee \dots \vee K_{i-1}$ in F and $C \vee K_1 \vee \dots \vee K_n$ is blocked in F.*

Example 40. *Consider the formula $F = \{\bar{P}(y) \vee R(y),\ \bar{R}(z) \vee Q(z)\}$ and the clause $C = P(a) \vee \bar{Q}(a)$. Although C is not blocked in F, we can add the literal $R(a)$ since it is contained in its only $P(a)$-resolvent, obtained by resolving with $\bar{P}(y) \vee R(y)$. The resulting clause $P(a) \vee \bar{Q}(a) \vee R(a)$ is then blocked by $R(a)$ as there is only the tautological $R(a)$-resolvent $P(a) \vee \bar{Q}(a) \vee Q(a)$, obtained by resolving with $\bar{R}(z) \vee Q(z)$. Thus, C is covered in F.*

Theorem 45. *If a clause C is covered in a formula F, it is redundant with respect to F.*

Proof. Suppose C is covered in F, meaning that we can add covered literals to C to obtain a clause $C \vee K_1 \vee \dots \vee K_n$ that is blocked in F. By Lemma 44, $F \wedge C$ and $F \wedge (C \vee K_1 \vee \dots \vee K_n)$ are equisatisfiable. Moreover, since $C \vee K_1 \vee \dots \vee K_n$ is blocked in F, it follows that F and $F \wedge (C \vee K_1 \vee \dots \vee K_n)$ are equisatisfiable. But then F and $F \wedge C$ are equisatisfiable and so C is redundant with respect to F. □

5.1.5 Resolution-Subsumed Clauses and More

The redundancy property of *resolution-subsumed clauses* (RS), which is used in SAT solving and which we already encountered briefly on page 33, can also be straightforwardly lifted to first-order logic, where redundancy is again an immediate consequence of Theorem 39 since subsumption ensures implication:

Definition 45. *A clause C is* resolution subsumed *in a formula F if it contains a literal L such that all non-tautological L-resolvents of C, with clauses in F, are subsumed in F.*

Note that resolution-subsumed clauses are different from clauses derived via the subsumption-resolution rule mentioned on page 5.1.3.

Theorem 46. *If a clause is resolution subsumed in a formula F, then it is redundant with respect to F.*

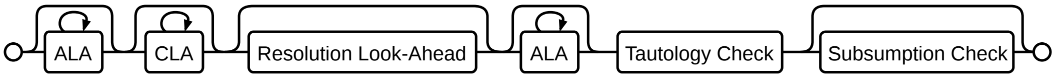


Figure 5.1: Combination of Techniques to Obtain Redundancy Properties.

With the methods presented so far, we can lift even more redundancy properties that have been considered in the SAT literature. We can do so by combining asymmetric-literal addition or covered-literal addition with tautology or subsumption checks. These checks can be performed either directly on the clause or for all resolvents of the clause upon one of its literals. The latter can be seen as some kind of "look-ahead" via resolution. Figure 5.1 illustrates possible combinations of techniques. Every path from the left to the right gives rise to a particular redundancy property. Remember that ALA stands for asymmetric-literal addition and CLA stands for covered-literal addition.

For instance, to detect if a clause is an asymmetric tautology, we first perform some asymmetric-literal additions and then check if the resulting clause is a tautology. Another example are blocked clauses, where we ask if all L-resolvents of the clause are tautologies. Similarly, we obtain covered clauses, resolution-subsumed clauses, and resolution asymmetric tautologies via such combinations. This gives rise to various other types of clauses like *asymmetric blocked clauses*, *asymmetric subsumed clauses* [JHB12], or *asymmetric covered clauses* [HJL$^+$15]. The redundancy of these clauses follows from the results in this chapter, most importantly from the principle of implication modulo resolution.

5.1.6 Confluence Properties of Elimination Techniques

In this section, we consider clause-elimination techniques based on the previously lifted redundancy notions and analyze their confluence. We also analyze confluence properties of the corresponding literal-addition techniques. Intuitively, confluence of a technique tells us that the order in which we perform the clause eliminations or the literal additions is not relevant to the final outcome of the technique.

Our notion of redundancy from Definition 34 says that a clause C is redundant with respect to a formula F if F and $F \wedge C$ are equisatisfiable. This means that if a clause D is contained in F, we can safely eliminate D from F if D is redundant with respect to $F \setminus \{D\}$, since then F and $F \setminus \{D\}$ are guaranteed to be equisatisfiable. The redundancy properties of the previous sections therefore give rise to corresponding clause elimination techniques.

To analyze confluence of these techniques formally, we interpret them as *abstract reduction systems* [BN98]. For instance, to analyze the confluence of a clause-elimination technique CE, we define the (reduction) relation $\rightarrow_{\mathsf{CE}}$ over formulas as follows: $F_1 \rightarrow_{\mathsf{CE}} F_2$ if and only if the technique CE allows us to obtain F_2 from F_1 by removing a clause. Likewise, for a literal-addition technique LA, we define the relation $\rightarrow_{\mathsf{LA}}$ over clauses as $C_1 \rightarrow_{\mathsf{LA}} C_2$ if and only if the technique LA allows us to obtain C_2 from C_1 by adding a literal. Hence, when we ask if a certain preprocessing technique is confluent, what we actually want to know is whether its corresponding reduction relation is confluent [BN98]:

Definition 46. *Let $\rightarrow$ be a relation and $\rightarrow_*$ its reflexive transitive closure. Then, $\rightarrow$ is* confluent *if, for all x, y_1, y_2 with $x \rightarrow_* y_1$ and $x \rightarrow_* y_2$, there exists an element z such that $y_1 \rightarrow_* z$ and $y_2 \rightarrow_* z$.*

In our context, this means that whenever the elimination of certain clauses from a formula F yields a formula F_1, and the elimination of certain other clauses from F yields another formula F_2, then there is still a formula F_z that we can obtain from both F_1 and F_2. Likewise for the addition of literals to a clause. Therefore, if an elimination technique is confluent, we do not need to worry about "missed opportunities" caused by a bad choice of the elimination order. For some techniques in this thesis, we can show the stronger *diamond property*, which implies confluence [BN98]:

Definition 47. *A relation $\rightarrow$ has the* diamond property *if, for all x, y_1, y_2 with $x \rightarrow y_1$ and $x \rightarrow y_2$, there exists a z such that $y_1 \rightarrow z$ and $y_2 \rightarrow z$.*

We start with analyzing the confluence of blocked-clause elimination, which actually enjoys the diamond property. Define the relation $\rightarrow_{\mathsf{BCE}}$ over formulas as follows: $F \rightarrow_{\mathsf{BCE}} G$ iff $G = F \setminus \{C\}$ and C is blocked in G.

Theorem 47. *Blocked-clause elimination is confluent, i.e., $\rightarrow_{\mathsf{BCE}}$ is confluent.*

Proof. If a clause C is blocked in a formula F, it is also blocked in every subset G of F, since the L-resolvents of C with clauses in G are a subset of the L-resolvents with clauses in F. Therefore, if all L-resolvents of C with clauses in F are tautologies, so are those with clauses in G. Hence, the relation $\rightarrow_{\mathsf{BCE}}$ clearly has the diamond property and thus it is confluent. □

As in the propositional case, where covered-clause elimination is confluent [HJL+15], we can prove the confluence of its first-order variant. Define $F \rightarrow_{\mathsf{CCE}} G$ iff $G = F \setminus \{C\}$ and C is covered in G.

Theorem 48. *Covered-clause elimination is confluent, i.e., $\rightarrow_{\mathsf{CCE}}$ is confluent.*

Proof. We show that $\rightarrow_{\mathsf{CCE}}$ has the diamond property. Let F_1 be obtained from a formula F by removing a literal C that is covered in $F \setminus \{C\}$ and let F_2 be obtained from F by removing a literal D that is covered in $F \setminus \{D\}$. It suffices to prove that both C and D are covered in $F \setminus \{C, D\}$. We show that C is covered in $F \setminus \{C, D\}$. The other case is symmetric. Since C is covered in $F \setminus \{C\}$, we can perform a sequence of $n \geq 0$ covered-literal additions where every literal K_i is covered by $C_{i-1} = C \vee K_1 \vee \cdots \vee K_{i-1}$ in $F \setminus \{C\}$ and $C_n = C \vee K_1 \vee \cdots \vee K_n$ is blocked in $F \setminus \{C\}$.

Now, if in $F \setminus \{C, D\}$, the clause C_n can be obtained from C by performing the same sequence of covered-literal additions, then C_n is also blocked in $F \setminus \{C, D\}$, which implies that C is covered in $F \setminus \{C, D\}$. Assume now to the contrary that there exists a literal K_i that is not covered by C_{i-1} in $F \setminus \{C, D\}$ and suppose without loss of generality that K_i is the first such literal. It follows that there exists a non-tautological L-resolvent of C_{i-1} (with a clause in $F \setminus \{C, D\}$) that does not contain K_i. But then K_i is not covered by C_{i-1} in $F \setminus \{C\}$, a contradiction. □

Covered-literal addition is confluent as well. Let F be a formula and define $C_1 \rightarrow_{\mathsf{CLA}} C_2$ iff C_2 can be obtained from C_1 by adding a literal K that is covered by C_1 in F.

Theorem 49. *Covered-literal addition is confluent, i.e.,* $\rightarrow_{\mathsf{CLA}}$ *is confluent.*

Proof. We show that the relation $\rightarrow_{\mathsf{CLA}}$ has the diamond property. Let F be formula and C a clause. Let furthermore $C_1 = C \lor K_1$ and $C_2 = C \lor K_2$ be obtained from C by respectively adding literals K_1 and K_2 that are both covered by C in F. We have to show that C_1 covers K_2 and, analogously, that C_2 covers K_1. Since C covers K_2, it follows that C contains a literal L such that K_2 is contained in all non-tautological L-resolvents of C. But, as $L \in C_1$, every non-tautological L-resolvent of C_1 must also contain K_2. It follows that C_1 covers K_2. The argument for K_1 being covered by C_2 is symmetric. □

Asymmetric-literal addition is also confluent. Let F be a formula and define $C_1 \rightarrow_{\mathsf{ALA}} C_2$ iff C_2 can be obtained from C_1 by adding a literal L that is an asymmetric literal with respect to C_1 in F.

Theorem 50. *Asymmetric-literal addition is confluent, i.e.,* $\rightarrow_{\mathsf{ALA}}$ *is confluent.*

Proof. If L_1 is an asymmetric literal with respect to a clause C in a formula F, then there exists a clause $D \lor \bar{L} \in F$ and a substitution λ such that $D\lambda \subseteq C$ and $L_1 = \bar{L}\lambda$. Thus, $D\lambda \subseteq C \lor L_2$ for each $C \lor L_2$ that was obtained from C by adding some asymmetric literal L_2, and so L_1 is an asymmetric literal with respect to every such clause. Hence, $\rightarrow_{\mathsf{ALA}}$ has the diamond property and so it is confluent. □

For asymmetric-tautology elimination, the non-confluence result from propositional logic [HJL+15] implies non-confluence of the first-order generalization. Finally, the following example shows non-confluence for both the elimination of resolution-subsumed clauses (RS) and the elimination of resolution asymmetric tautologies (RAT):

Example 41. *Let $F = \{P \lor \bar{Q},\ Q \lor \bar{R},\ \bar{P} \lor R,\ \bar{Q} \lor R\}$. Then, $\bar{Q} \lor R$ is a* RAT *and* RS *on the literal R in $F \setminus \{\bar{Q} \lor R\}$ as there is only one R-resolvent, namely the tautology $\bar{Q} \lor Q$, obtained by resolving with $Q \lor \bar{R}$. If we remove $\bar{Q} \lor R$ from F, none of the remaining clauses of F is a* RAT *or* RS *with respect to the other remaining clauses. In contrast, suppose we start by removing $P \lor \bar{Q}$, which is a* RAT *and* RS *on P with respect to $F \setminus \{P \lor \bar{Q}\}$, then all the other clauses can afterwards be removed, because*

Technique	Confluent
Blocked-Clause Elimination	yes
Covered-Clause Elimination	yes
Asymmetric-Tautology Elimination	no
Resolution-Asymmetric-Tautology Elimination	no
Resolution-Subsumed-Clause Elimination	no
Covered-Literal Addition	yes
Asymmetric-Literal Addition	yes

Table 5.1: Confluence Properties of First-Order Clause-Elimination Techniques.

they become blocked with respect to the other clauses, implying that they are RAT *and* RS*: The clause $\bar{R} \lor Q$ becomes blocked by the literal Q as there is only a tautological resolvent upon Q, namely $\bar{R} \lor R$. For $\bar{P} \lor R$, there are no resolvents upon $\bar{P}$ and so it trivially becomes blocked by $\bar{P}$. Finally, $\bar{Q} \lor R$ becomes blocked by both R and $\bar{Q}$ as there are only tautological resolvents upon these two literals.*

A summary of the confluence results is given in Table 5.1.

5.2 First-Order Logic With Equality

All the previously mentioned redundancy properties only guarantee redundancy in first-order logic *without* equality. In the following, we present a variant of implication modulo resolution that also guarantees redundancy in first-order logic *with* equality. We obtain first-order logic with equality by adding a distinct predicate symbol $\approx$ that must be interpreted by the identity relation over the domain under consideration.

It is well known that if we consider a set $\mathcal{E}_\mathcal{L}$ of *equality axioms* (see below), then a formula F that contains the equality predicate is satisfiable if and only if $F \cup \mathcal{E}_\mathcal{L}$ is satisfiable without the restriction that $\approx$ must be interpreted as the identity relation. The equality axioms $\mathcal{E}_\mathcal{L}$ denote the following set of clauses for the language $\mathcal{L}$ under consideration (we write $x \not\approx y$ to denote a literal of the form $\neg x \approx y$) :

(E1) $x \approx x$;

(E2) $x \not\approx y \lor y \approx x$;

(E3) $x \not\approx y \lor y \not\approx z \lor x \approx z$;

(E4) for each n-ary function symbol f in $\mathcal{L}$,
$x_1 \not\approx y_1 \lor \cdots \lor x_n \not\approx y_n \lor f(x_1, \dots, x_n) \approx f(y_1, \dots, y_n)$;

(E5) for each n-ary predicate symbol P in $\mathcal{L}$,
$x_1 \not\approx y_1 \lor \cdots \lor x_n \not\approx y_n \lor \bar{P}(x_1, \dots, x_n) \lor P(y_1, \dots, y_n)$.

The previous variant of Herbrand's Theorem (Theorem 37) does not hold in the presence of equality, but the following variant does:

Theorem 51. *A formula F that contains the equality predicate is satisfiable if and only if every finite set of ground instances of clauses in $F \cup \mathcal{E}_{\mathcal{L}}$ is propositionally satisfiable.*

In the following, we introduce a variant of implication modulo resolution, called *implication modulo flat resolution*, that ensures redundancy in first-order logic with equality. We then show how implication modulo flat resolution gives a short correctness proof of the predicate-elimination technique by Khasidashvili and Korovin [KK16]. We also introduce so-called *equality-blocked clauses*, a variant of blocked clauses that is redundant in first-order logic with equality. Finally, we discuss and evaluate a practical application of blocked clauses, namely *blocked-clause elimination* as a preprocessing step for first-order theorem provers. Defining appropriate equality variants for other redundancy properties such as covered clauses and resolution asymmetric tautologies is part of our future work.

5.2.1 Implication Modulo Flat Resolution

Implication modulo resolution ensures redundancy in first-order logic without equality. In the presence of equality, however, redundancy is not guaranteed:

Example 42. *Let $C = P(a)$ and $F = \{a \approx b, \bar{P}(b)\}$. Since $P(a)$ and $P(b)$ are not unifiable, there are no resolvents of C, hence F trivially implies C modulo resolution. But, F is clearly satisfiable whereas $F \wedge C$ is not.*

In the example, models of F must assign the same truth value to $P(a)$ and $P(b)$. Hence, when trying to turn a model of F into one of $F \wedge C$ by flipping the truth value of $P(a)$, we also need to flip the truth value of $P(b)$ although $P(a)$ and $P(b)$ are not unifiable.

Thus, to ensure redundancy in the presence of equality, it is not enough to consider only the clauses that are L-resolvable with C. We need to take all clauses that contain a literal of the form $\bar{L}(\dots)$ into account. To do so, we make use of *flattening* [CP86]:

Definition 48. *Let $C = L(t_1, \dots, t_n) \vee C'$ be a clause.* Flattening *the literal $L(t_1, \dots, t_n)$ in C yields the clause $C^- = \bigvee_{1 \leq i \leq n} x_i \not\approx t_i \vee L(x_1, \dots, x_n) \vee C'$, with $x_i, \dots, x_n$ being fresh variables.*

Example 43. *Flattening the literal $P(f(x), c, c)$ in the clause $P(f(x), c, c) \vee Q(c)$ yields the new clause $x_1 \not\approx f(x) \vee x_2 \not\approx c \vee x_3 \not\approx c \vee P(x_1, x_2, x_3) \vee Q(c)$.*

To see that flattening preserves equivalence, observe that the clause resulting from flattening $L(t_1, \dots, t_n)$ in $L(t_1, \dots, t_n) \vee C'$ is equivalent to an implication of the form $(x_1 \approx t_1 \wedge \dots \wedge x_n \approx t_n) \rightarrow (L(x_1, \dots, x_n) \vee C')$. We can now define *flat resolvents*, which are obtained by first flattening literals and then resolving upon them, which allows us to resolve upon literals that are otherwise not unifiable:

Definition 49. *Let $C = L \lor C'$ and $D = N_1 \lor \cdots \lor N_k \lor D'$ (with $k > 0$) be clauses such that the literals $L, \bar{N}_1, \ldots, \bar{N}_k$ have the same predicate symbol and polarity. Let furthermore C^- and D^- be obtained from C and D, respectively, by flattening $L, N_1, \ldots, N_k$, and denote the flattened literals by $L^-, N_1^-, \ldots, N_k^-$. The resolvent*

$$(C^- \setminus \{L^-\})\sigma \;\lor (D^- \setminus \{N_1^-, \ldots, N_k^-\})\sigma$$

of C^- and D^-, with σ being an mgu of $L^-, \bar{N}_1^-, \ldots, \bar{N}_k^-$, is a flat L-resolvent *of C and D.*

Note that after flattening, the literals $L^-, \bar{N}_1^-, \ldots, \bar{N}_k^-$ are of the form $L(x_1, \ldots, x_n)$, $\bar{N}_1(y_{11}, \ldots, y_{1n})$, $\ldots$, $\bar{N}_k(y_{k1}, \ldots, y_{kn})$, respectively. As these literals contain only variables, they are easily unified by the unifier $\bigcup_{i=1}^{k}\{y_{ij} \mapsto x_j \mid 1 \leq j \leq n\}$, which maps every variable at the j-th position to the variable x_j. This unifier is in fact a most general unifier (c.f. [BN98]). We will make use of this fact later on in the proof of Lemma 52. The following example illustrates flat resolvents:

Example 44. *Consider again the clause $C = P(a)$ and the formula $F = \{a \approx b, \bar{P}(b)\}$ from Example 42 and let $D = P(b)$. By flattening $P(a)$ in C and $\bar{P}(b)$ in D we obtain $C^- = x_1 \not\approx a \lor P(x_1)$ and $D^- = y_1 \not\approx b \lor \bar{P}(y_1)$, respectively. Their resolvent $x_1 \not\approx a \lor x_1 \not\approx b$ is a flat $P(a)$-resolvent of C and D.*

Using flat resolution, we can now define the principle of *implication modulo flat resolution*, which guarantees redundancy even in first-order logic with equality:

Definition 50. *A clause C is* implied modulo flat resolution *by a formula F if it contains a literal L such that the predicate of L is not $\approx$ and all flat L-resolvents of C, with clauses in F, are implied by F.*

To prove the redundancy of clauses that are implied modulo flat resolution, we first define the notion of *equivalence flipping*. Intuitively, equivalence flipping of a ground literal $L(t_1, \ldots, t_n)$ turns a propositional assignment α into an assignment α' by inverting not only the truth value of $L(t_1, \ldots, t_n)$ but also the truth values of all literals $L(s_1, \ldots, s_n)$ for which α satisfies the equalities $t_1 \approx s_1, \ldots, t_n \approx s_n$.

Definition 51. *Let α be a propositional assignment and let $L(t_1, \ldots, t_n)$ be a ground literal with predicate symbol P other than $\approx$. The assignment α', obtained by* equivalence flipping *the truth value of $L(t_1, \ldots, t_n)$, is defined as follows:*

$$\alpha'(A) = \begin{cases} 1 - \alpha(A) & \text{if } A = P(s_1, \ldots, s_n) \text{ and } \;\alpha(t_i \approx s_i) = 1 \text{ for all } 1 \leq i \leq n, \\ \alpha(A) & \text{otherwise.} \end{cases}$$

Clearly, equivalence flipping preserves the truth of instances of the equality axioms. We can now prove Lemma 52 below, which is the equality-variant of Lemma 38 on page 73.

Lemma 52. *Let C be implied modulo flat resolution upon L by F, and let α be a propositional assignment that satisfies all ground instances of $F \cup \mathcal{E}_{\mathcal{L}}$ but falsifies a ground instance $C\lambda$ of C. Then, the assignment α', obtained from α by equivalence flipping the truth value of $L\lambda$, still satisfies all ground instances of clauses in $F \cup \mathcal{E}_{\mathcal{L}}$.*

Proof. Let $L = L(t_1, \dots, t_n)$ and $C = L \vee C'$ and suppose α falsifies a ground instance $C\lambda$ of C. Now obtain α' from α by equivalence flipping the truth value of $L(t_1, \dots, t_n)\lambda$. Since equivalence flipping does not affect the equality axioms, α' could only possibly falsify clauses of the form $D\tau$ where $D \in F$ and $\bar{L}(s_1, \dots, s_n)\tau \in D\tau$ such that $\alpha(t_i\lambda \approx s_i\tau) = 1$ for $1 \leq i \leq n$.

Let $D\tau$ be such a clause and let $\bar{L}(s_1, \dots, s_n), \dots, \bar{L}(r_1, \dots, r_n)$ be the literals of D such that α satisfies the equalities $t_i\lambda \approx s_i\tau, \dots, t_i\lambda \approx r_i\tau$ for $1 \leq i \leq n$. To simplify the presentation, we assume that $\bar{L}(s_1, \dots, s_n)$ and $\bar{L}(r_1, \dots, r_n)$ are the only such literals (for another number of such literals, the proof is analogous). We show that α' satisfies $D\tau$. First, observe that D is of the form $\bar{L}(s_1, \dots, s_n) \vee \bar{L}(r_1, \dots, r_n) \vee D'$.

As F implies C modulo flat resolution upon $L(t_1, \dots, t_n)$, all flat $L(t_1, \dots, t_n)$-resolvents of C are implied by F. Therefore, the particular flat $L(t_1, \dots, t_n)$-resolvent

$$R = (C' \vee D' \vee \bigvee_{1 \leq i \leq n} x_i \not\approx t_i \vee y_i \not\approx s_i \vee z_i \not\approx r_i)\sigma$$

is implied by F, where σ is an *mgu* of the literals $L(x_1, \dots, x_n)$, $L(y_1, \dots, y_n)$, and $L(z_1, \dots, z_n)$, which were obtained by respectively flattening $L(t_1, \dots, t_n)$, $\bar{L}(s_1, \dots, s_n)$, and $\bar{L}(r_1, \dots, r_n)$. Assume w.l.o.g. that $\sigma = \{y_i \mapsto x_i \mid 1 \leq i \leq n\} \cup \{z_i \mapsto x_i \mid 1 \leq i \leq n\}$. Then,

$$R = C' \vee D' \vee \bigvee_{1 \leq i \leq n} x_i \not\approx t_i \vee x_i \not\approx s_i \vee x_i \not\approx r_i.$$

As R is implied by F, the assignment α must satisfy all ground instances of R. Consider therefore the following substitution γ that yields a ground instance $R\gamma$ of R:

$$\gamma(x) = \begin{cases} t_i\lambda & \text{if } x \in \{x_1, \dots, x_n\}, \\ x\lambda & \text{if } x \in var(C), \\ x\tau & \text{if } x \in var(D). \end{cases}$$

We observe that the ground instance $R\gamma$ of R is the clause

$$C'\lambda \vee D'\tau \vee \bigvee_{1 \leq i \leq n} t_i\lambda \not\approx t_i\lambda \vee t_i\lambda \not\approx s_i\tau \vee t_i\lambda \not\approx r_i\tau$$

which must be satisfied by α. Now, the inequalities of the form $t_i\lambda \not\approx t_i\lambda$ are clearly falsified by α. Furthermore, by assumption, α falsifies all the inequalities $t_i\lambda \not\approx s_i\tau$ and $t_i\lambda \not\approx r_i\tau$ as well as $C'\lambda$. But then α must satisfy at least one of the literals in $D'\tau$. Since none of the literals in $D'\tau$ can be affected by equivalence flipping the truth value of $L\lambda$ (as D' does not contain a literal of the form $\bar{L}(\dots)$), $D'\tau$ must be satisfied by α'. It follows that α' satisfies $D\tau$. $\square$

Using Lemma 52 instead of Lemma 38, and using the equality variant of Herbrand's Theorem (Theorem 51), the proof of redundancy for implication modulo flat resolution is analogous to the proof of Theorem 39. We thus get:

Theorem 53. *If a formula F implies a clause C modulo flat resolution, then C is redundant with respect to F.*

5.2.2 Predicate Elimination

The principle of implication modulo flat resolution allows us to construct a short soundness proof for the predicate-elimination technique of Khasidashvili and Korovin [KK16]. Predicate elimination is a first-order variant of variable elimination, which is successfully used during preprocessing and inprocessing in SAT solving [EB05].

The elimination of a predicate P from a formula F is computed as follows: First, we add to F all flat resolvents upon literals with predicate symbol P. After this, we remove all original clauses that contain P. To guarantee that this procedure does not affect satisfiability, Khasidashvili and Korovin require P to be *non-self-referential*, meaning that it must not occur more than once per clause. Note that their notion of a self-referential predicate differs from our notion of a *recursive* literal, which we used in the context of covered clauses. The following statement holds in first-order logic with equality:

Theorem 54. *If a formula G is obtained from a formula F by eliminating a non-self-referential predicate P, then F and G are equisatisfiable.*

Proof. Let F_P be obtained from F by adding all flat resolvents upon P. Clearly, F_P and F are equivalent. Now, let C be a clause that contains a literal L with predicate symbol P. Then, F_P contains all flat L-resolvents of C with clauses in $F_P \setminus \{C\}$, which means that $F_P \setminus \{C\}$ implies C modulo flat resolution. We can thus remove C from F_P without affecting satisfiability. Hence, we can remove all clauses that contain the predicate P until we obtain the formula G. We conclude that F and G are equisatisfiable. $\square$

As mentioned by Khasidashvili and Korovin, the negative equalities (i.e., equalities of the form $x \not\approx t$) introduced by flattening can be eliminated again afterwards, using the equivalence-preserving rule of *equality substitution*. Equality substitution replaces a clause of the form $(C \vee x \not\approx t)$ by the clause $C[t/x]$, obtained from C by replacing all occurrences of x by t.

We want to highlight that this variant of predicate elimination is sound in first-order logic with equality. In first-order logic without equality, we can avoid the flattening and just add ordinary binary resolvents. The soundness proof is analogous to the one above, using implication modulo resolution instead of implication modulo flat resolution.

5.2.3 Equality-Blocked Clauses

The principle of implication modulo flat resolution allows us to define a blocked-clause notion that guarantees redundancy in the presence of equality; we call these clauses *equality-blocked clauses* (note that here we require resolvents to be *valid* instead of being *tautological*, thereby also including clauses such as $x \not\approx y \lor x \not\approx z \lor y \approx z$):

Definition 52. *A clause C is* equality-blocked *in a formula F if C contains a literal L such that the predicate of L is not $\approx$ and all flat L-resolvents of C with clauses in F are valid.*

We say that C is equality-blocked *by* L in F. Note that a flat L-resolvent R is valid if and only if the negation of its universal closure $\neg\forall x_1 \dots \forall x_n R$, where $x_1, \dots x_n$ are the variables occurring in R, is unsatisfiable. Skolemization (which introduces fresh constants for the variables of R) turns the formula $\neg\forall x_1 \dots \forall x_n R$ into a conjunction of ground (equational) literals, which can be efficiently decided by a congruence-closure algorithm like the one by Shostak [Sho78].

Since valid clauses are trivially implied, equality-blocked clauses are implied modulo flat resolution. We thus get:

Theorem 55. *If a clause is equality-blocked in a formula F, it is redundant with respect to F.*

The following example stems from a first-order encoding of an AI-benchmark problem known as "Who killed Aunt Agatha?" [Pel86]:

Example 45. *Consider the formula $F = \{L(a), L(b), L(c), \bar{L}(x) \lor x \approx a \lor x \approx b \lor x \approx c\}$. Intuitively, the clauses $L(a)$, $L(b)$, and $L(c)$ encode that there are three living individuals: Agatha, Butler, and Charles. The clause $\bar{L}(x) \lor x \approx a \lor x \approx b \lor x \approx c$ encodes that these three individuals are the only living individuals. We can observe that all four clauses are equality-blocked with respect to the other clauses of F. For instance, let $C = L(a)$. There exists one flat $L(a)$-resolvent of C: the valid clause $x_1 \not\approx a \lor x_1 \not\approx x \lor x \approx a \lor x \approx b \lor x \approx c$, obtained by resolving the clauses $x_1 \not\approx a \lor L(x_1)$ and $y_1 \not\approx x \lor \bar{L}(y_1) \lor x \approx a \lor x \approx b \lor x \approx c$.*

Finally, note that regarding confluence, the argument showing that blocked-clause elimination is confluent carries over to equality-blocked clauses.

5.3 Blocked-Clause Elimination in Practice

In this section, we present the implementation and evaluation of a first-order preprocessing tool that performs blocked-clause elimination (BCE). We further discuss how BCE eliminates pure predicates and how it is related to the existing preprocessing technique of *unused-definition elimination* (UDE) by Hoder et al. [HKKV12].

5.3.1 Implementation

We implemented blocked-clause elimination for first-order logic as a preprocessing step in the automated theorem prover Vampire [KV13].[1] Depending on whether or not the formula at hand contains the equality predicate, Vampire performs either the elimination of equality-blocked clauses or the elimination of blocked clauses. The elimination is performed as the last step in the preprocessing pipeline, because it relies on the input being in CNF. After the preprocessing, instead of proceeding to proving the formula—which is the default behavior—Vampire can be instructed to output the final set of clauses.

The top level organization of our elimination procedure is inspired by the approach adopted in the propositional case by Järvisalo et al. [JBH10]. For efficiency, we maintain an index for accessing a literal within a clause by its predicate symbol and polarity. The main data structure is a priority queue of candidates (L, C) where C is a clause that is potentially blocked by the literal L in the formula under consideration. We prioritize for processing those candidates (L, C) which have fewer potential resolution partners, estimated by the number of clauses indexed with the same predicate symbol and the opposite polarity as L.

At the beginning, every (non-equational) literal L in a clause C gives rise to a candidate (L, C). We always pick the next candidate (L, C) from the queue and iterate over potential resolution partners D. If we discover that a (flat) L-resolvent of C and D is not valid, further processing of (L, C) is postponed and the candidate is "remembered" by the partner clause D. If, on the other hand, all the (flat) L-resolvents with all the possible partners D have been found valid, the clause C is declared blocked and the candidates remembered by C are "resurrected" and put back to the queue. Their processing will be resumed by iterating over those partners which have not been tried yet.

Our implementation uses for efficiency reasons an approximate solution which only computes binary (flat) resolvents. Then, before testing a binary resolvent for validity, we remove from it all literals that (1) are unifiable with $\bar{L}\sigma$ in the blocking case, or (2) have the same predicate symbol and polarity as $\bar{L}$ in the equality-blocking case. This still ensures redundancy and significantly improves the performance.

For testing validity of flat L-resolvents in the equality case, we experimented with a complete congruence-closure procedure which turned out to be too inefficient. Our current implementation only "normalizes" in a single pass all (sub-)terms of the literals in a flat resolvent using the equations from the flattening, but it ignores (dis-)equations originally present in the two clauses and it does not employ the congruence rule recursively. Our experiments show that even this limited version is effective.

[1] A statically compiled x86_64 executable of Vampire used in our experiments can be obtained from `http://forsyte.at/wp-content/uploads/vampire_bce.zip`.

5.3.2 Relation to Existing Preprocessing Techniques

In the propositional setting, blocked-clause elimination is known to simulate on the CNF-level several refinements of the standard CNF encoding for circuits [JBH10]. Similarly, we observe that in the first-order setting BCE simulates *pure-predicate elimination* (PPE) and we conjecture that under certain conditions (discussed later) it also simulates *unused-definition elimination* (UDE), a formula-level simplification as described by Hoder et al. [HKKV12]. In this section we briefly discuss these two techniques and explain their relation to BCE. Apart from being of independent interest, the observations made in this section are also relevant for interpreting the experimental results presented later.

We say that a predicate symbol P is *pure* in a formula F if, in F, all occurrences of literals with predicate symbol P are of the same polarity. If a clause C contains a literal L with a pure predicate symbol P, then there are no L-resolvents of C, hence C is vacuously blocked. Therefore, blocked-clause elimination removes all clauses that contain pure predicates and thus it simulates pure-predicate elimination.

UDE is a preprocessing method that removes so-called *unused predicate definitions* from formulas that are not necessarily in CNF. Given a predicate symbol P and a general formula φ such that P does not occur in φ, a *predicate definition* is a formula

$$def(P, \varphi) = \forall x_1 \ldots \forall x_n \; P(x_1, \ldots, x_n) \leftrightarrow \varphi(x_1, \ldots, x_n).$$

Assuming we have a predicate definition $def(P, \varphi)$ as a conjunct within a larger formula of the form $\Psi = \psi \wedge def(P, \varphi)$, the definition is *unused* in Ψ if P does not occur in ψ. UDE allows the elimination of such a definition and it is guaranteed that this elimination preserves unsatisfiability [HKKV12]. (In fact, if P only occurs in ψ with a single polarity, then one of the two implications of the equivalence $def(P, \varphi)$, corresponding to that polarity, can be dropped by UDE.)

Note that UDE operates on the level of general formulas while BCE is only defined for formulas in CNF. Let therefore $def(P, \varphi)$ be an unused predicate definition in the formula $\Psi = \psi \wedge def(P, \varphi)$ as above and let $BCE(\mathsf{cnf}(\Psi))$ be the result of eliminating all blocked clauses from a clause form translation $\mathsf{cnf}(\Psi)$ of Ψ. We conjecture that for any “reasonably behaved” clausification procedure cnf (like, e.g., the well-known Tseitin encoding [Tse68]), it holds that $BCE(\mathsf{cnf}(\Psi)) \subseteq \mathsf{cnf}(\psi)$ if φ does not contain quantifiers. In other words, BCE simulates UDE under the above conditions.

The main idea behind the simulation would be to show that each clause stemming from the clausification of an unused definition $def(P, \varphi)$ is blocked on the literal that corresponds to the predicate P. The reason why the presence of quantifiers in the definition formula φ poses a problem can be highlighted on the following simple example:

Example 46. *The predicate definition $def(P, \exists x Q(x)) = P \leftrightarrow \exists x Q(x)$ can be clausified as $\bar{P} \vee Q(c)$, $P \vee \bar{Q}(x)$, where c is a Skolem constant corresponding to the existential quantifier. By resolving these two clauses on P we obtain the resolvent $Q(c) \vee \bar{Q}(x)$, which is not valid.*

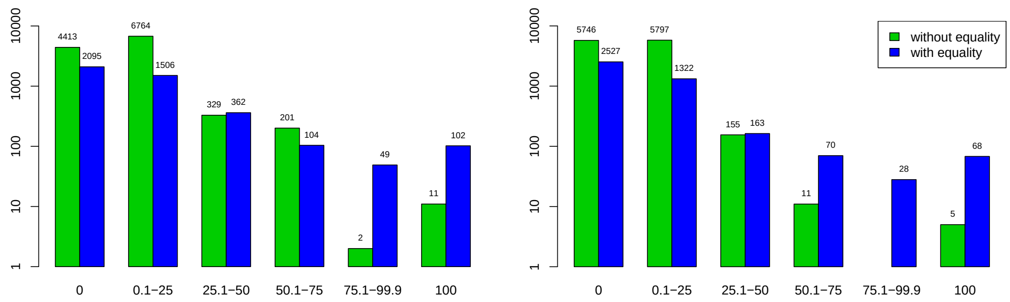


Figure 5.2: Histogram of problems by the percentage of blocked clauses eliminated. BCE applied after simple clausification (left) and after clausification preceded by PPE and UDE (right).

5.3.3 Empirical Evaluation

We present an empirical evaluation of our implementation of blocked-clause elimination and equality-blocked-clause elimination, which is part of the preprocessing pipeline of the automated theorem prover Vampire [KV13]. In our experiments, we used the 15 942 first-order benchmark formulas of the TPTP library [Sut09] (version 6.4.0).[2]

Of these benchmarks, 7898 were already in CNF, while the remaining 8044 general formulas needed to be clausified by Vampire before being subjected to our clause elimination procedures. This clausification step was optionally preceded by Vampire's implementation of PPE and UDE (see Section 5.3.2). 73 % of the benchmark formulas contain the equality predicate. In these formulas, we eliminated equality-blocked clauses while in the others we eliminated blocked clauses. All experiments were run on the StarExec compute cluster [SST14].[3]

Occurrence of Blocked Clauses. Within a time limit of 300 s for parsing, clausification (if needed), and subsequent blocked-clause detection and elimination, our implementation was able to process all but one problem. The average/median time for detecting and eliminating blocked clauses was 0.238 s/0.001 s.

In total, the benchmarks correspond to 299 379 591 clauses. BCE removes 11.72 % of these clauses, while independently processing the problems with PPE and UDE before clausification leads to 7.66 % fewer clauses. Combining both methods yields a total reduction of 11.73 %. Hence, the number of clauses which can be effectively removed by

[2]The TPTP library version 6.4.0 can can be downloaded at `http://tptp.cs.miami.edu/TPTP/Archive/TPTP-v6.4.0.tgz`.

[3] See `http://forsyte.at/static/people/suda/bce_starexec_solvers.zip` for the configurations of solvers used in our experiment.

Table 5.2: Effect of blocked-clause elimination on theorem proving strategies. Bold: numbers of solved problems without blocked-clause elimination; positive (negative): problems gained (lost, respectively) by using blocked-clause elimination.

	Unsatisfiable			Satisfiable			Total		
Vampire	**3172**	−28	+40	**458**	−0	+5	**3630**	−28	+45
E	**3097**	−20	+27	**363**	−1	+9	**3460**	−21	+36
CVC4	**2930**	−18	+37	**9**	−0	+68	**2939**	−18	+105

UDE but not by BCE or which can only be removed by BCE after some other clauses have been effectively removed by UDE is in the order of 0.01 %.

Out of the 15 941 benchmarks, 59 % contain a blocked clause after simple clausification and 48 % of these benchmarks contain a blocked clause if first processed by PPE and UDE. Figure 5.2 shows the detailed distribution of eliminated blocked clauses. With PPE and UDE disabled, more than 25 % of the clauses could be eliminated in over 1000 problems. Moreover, 113 satisfiable formulas were directly solved by BCE, which means that BCE rendered the input empty. After applying PPE and UDE, which directly solve 46 problems, subsequent BCE can directly solve 73 other problems. There are two problems which can only be directly solved by the combination of PPE, UDE, and BCE.

Impact on Proving Performance. To measure the effect of BCE on recent theorem provers, we considered the three best different systems of the main FOF division of the 2016 CASC competition [SU16]: Vampire 4.0, E 2.0, and CVC4 1.5.1.[4] Instead of running the provers in competition configurations, which are in all three cases based on a portfolio of strategies and thus lead to results that tend to be hard to interpret (cf. also the discussion given by Reger et al. [RSV14]), we asked the respective developers to provide a single representative strategy good for proving theorems by their prover and then used these strategies in the experiment.

We combined Vampire as a clausifier with the three individual provers using the Unix pipe construct. The clausification included PPE and UDE (enabled by default in Vampire) and either did or did not include BCE. We set a time limit of 300 s for the whole combination, so the possible time overhead incurred by BCE left less time for actual proving. We ran the systems on the 7619 problems established above on which BCE eliminates at least one clause.

Table 5.2 shows the numbers of solved problems without BCE and the difference when BCE is enabled. We can see that on satisfiable problems, BCE allows every prover to find more solutions; the most notable gain is observed with CVC4. BCE also enables

[4] Actually, Vampire 4.1 was ranked second, but we did not include it, as it is just an updated version of Vampire 4.0.

Table 5.3: Effect of blocked-clause elimination on satisfiability checking strategies. Bold: numbers of solved problems without blocked-clause elimination; positive (negative): problems gained (lost, respectively) by using blocked-clause elimination.

	Satisfiable			Unsatisfiable			Total		
Vampire	**531**	−0	+24	**719**	−4	+5	**1250**	−4	+29
iProver	**558**	−0	+1	**755**	−6	+4	**1313**	−6	+5
CVC4	**489**	−1	+28	**1724**	−24	+20	**2213**	−25	+48

each prover to solve new unsatisfiable problems, but there are problems that cannot be solved anymore (with the preselected strategy) when BCE is activated. Although the overall trend is that using BCE pays off, the existence of the lost problems is slightly puzzling. For a majority of them, the time taken to perform BCE is negligible and thus cannot explain the phenomenon. Moreover, proofs that would make use of a blocked clause, although they do sometimes occur, are quite rare.[5] Our current explanation thus appeals to the inherently “fragile” nature of the search spaces traversed by a theorem prover, in which the presence of a clause can steer the search towards a proof even if the clause does not itself directly take part in the proof in the end.

Strategies for Showing Satisfiability. Since the previous experiment indicates that BCE can be especially helpful on satisfiable problems, we decided to test how much it could improve strategies explicitly designed for establishing satisfiability, such as finite-model finding. This should be contrasted with the previous strategies, which focused on proving theorems. Here, we selected three systems successful in the FNT (First-order form Non-Theorems) division of the 2016 CASC competition, namely Vampire 4.1, iProver 2.5, and CVC4 1.5.1 and again picked representative strategies for each, this time focusing on satisfiability detection. The overall setup remained the same, with a time limit of 300 s.

Table 5.3 shows the results of this experiment. As we can see, Vampire and CVC4 solved significantly more satisfiable problems when BCE was used. On the other hand, iProver solved only one additional problem with the help of BCE. The results on unsatisfiable problems, which are not specifically targeted by the selected strategies, did not show a clear advantage of BCE.

Mock Portfolio Construction. Understanding the value of a new technique within a theorem prover is very hard. The reason is that—in its most powerful configuration—a theorem prover usually employs a portfolio of strategies and each of these strategies may

[5] For 51 of the 3172 problems shown to be unsatisfiable by Vampire, the corresponding proof contained a blocked clause. Nevertheless, none of these problems were among the 28 which Vampire did not solve after applying BCE.

respond differently to the introduction of a new technique. In fact, a portfolio constructed without regard to the new technique is most likely suboptimal because the new technique may—due to interactions which are typically hard to predict—give rise to new successful strategies that could not be considered previously [RSV14]. In this final experiment, which was carried out for us by Andrei Voronkov (the main designer of Vampire), we tried to establish the value of BCE for the construction of a new strategy portfolio in Vampire by emulating the typical first phase of the portfolio construction process, namely random sampling of the space of all strategies. Encouraged by the previous experiment, we focused on the construction of a portfolio specialized on detecting satisfiable problems.

The experiment involved 302 satisfiable problems from the TPTP library that were previously established hard for Vampire, and that all contain at least one predicate that is different from equality. The experiment was performed using randomly generated strategies by flipping values of various options that define how the prover attempts to establish satisfiability. Each strategy was cloned into two, one running with BCE as part of the preprocessing and the other without. Every such pair of strategies was then run on a randomly selected hard problem with a time limit of 120 s. In total, 50 000 pairs of strategies were considered.

Strategies using BCE succeeded 8414 times while strategies not using BCE succeeded 6766 times. In particular, there were 1796 cases where only the BCE variation succeeded on a problem compared to 148 cases where only the strategy without BCE succeeded. This demonstrates that BCE is a valuable addition to the set of Vampire options and so it will likely be employed by a considerable fraction, if not all, of the strategies of the satisfiability checking CASC mode portfolio of Vampire in the future.

CHAPTER 6

QRAT Simulates Long-Distance Resolution

After having dealt extensively with propositional logic and first-order logic, we finally arrive at quantified Boolean formulas (QBFs) [KB09], which extend propositional formulas with existential and universal quantifiers over the propositional variables. These quantifiers lead to succinct problem encodings, which makes QBF an attractive formalism for reasoning in areas such as formal verification and artificial intelligence [BM08].

So far, we have modified formulas in propositional logic and first-order logic mostly by adding and eliminating redundant clauses. Especially blocked clauses have played an important role throughout the previous chapters. In this chapter, we deal with a QBF concept that is closely related to blocked clauses—so-called *blocked literals*. We use both blocked-literal addition and blocked-literal elimination to clarify the relationship between two proof systems for QBF.

Proof systems for QBF have been extensively analyzed in order to obtain a better understanding of the strengths and limitations of different QBF-solving approaches. This has led to a comprehensive proof-complexity landscape, containing various proof systems that are very different [BCJ15, Egl16, Che16, BP16, Jan16]. Two kinds of proof systems have received particular attention: instantiation-based proof systems [BCJ14, BCJ15], which provide the foundation for expansion-based solvers like `RAReQS` [JKMSC16], and resolution-based proof systems [Jan16, KKF95, ZM02, BJ12, VG12a, BWJ14, BCMS16, JGMS13, SS14], which provide the foundation for search-based solvers like `DepQBF` [LE17]. Apart from these, also sequent systems have been studied [Egl16, BP16]. There is, however, another practically useful proof system—quite different from the aforementioned ones—whose place in the complexity landscape was still unclear: the QRAT proof system [HSB16].

The QRAT proof system is a generalization of DRAT, which we discussed in Chapter 3, where we dealt with propositional proof systems. One of the strengths of QRAT is its ability to simulate preprocessing techniques: Many QBF solvers use preprocessing techniques to simplify a QBF before they actually evaluate its truth. With the QRAT system, it is possible to certify the correctness of virtually all preprocessing simplifications performed by state-of-the-art QBF solvers and preprocessors. Additionally, there exist efficient tools for checking the correctness of QRAT proofs as well as for extracting winning strategies (so-called *Skolem functions*) from QRAT proofs of satisfiability [HSB16].

It can be easily seen that QRAT polynomially simulates the basic Q-resolution calculus [KKF95], meaning that there exists a polynomial-time procedure that transforms valid Q-resolution proofs into valid QRAT proofs. Likewise, QRAT polynomially simulates the calculus QU-Res [VG12a], which extends Q-resolution by allowing resolution upon universal variables. So far, however, it was unclear how QRAT is related to the long-distance-resolution calculus [ZM02, BJ12]—a calculus that is particularly popular because it allows for short proofs both in theory and in practice [ELW13].

In the remainder of this chapter, we prove that QRAT can polynomially simulate the long-distance-resolution calculus. For our simulation, we need only Q-resolution and universal reduction together with blocked-literal elimination and blocked-literal addition using fresh variables [HSB15, Kul99]. These four rules are allowed in QRAT. To illustrate the power of blocked literals, we present handcrafted QRAT proofs of the formulas commonly used to display the strength of long-distance resolution—the well-known *Kleine Büning formulas* [KKF95]. Our proofs are slightly smaller than the long-distance resolution proofs of these formulas described by Egly et al. [ELW13].

To put our simulation into practice, we implemented a tool that transforms long-distance-resolution proofs into QRAT proofs. With this tool it is now possible to obtain QRAT proofs that certify the correctness of both the preprocessing and the actual solving, even when using a QBF solver based on long-distance resolution. We used our tool to transform long-distance-resolution proofs of the Kleine Büning formulas into QRAT proofs. We compare the resulting proofs with the handcrafted QRAT proofs as well as with the original proofs. Rounding off the picture, we relate QRAT to popular resolution-based proof systems and discuss open questions.

6.1 Quantified Boolean Formulas

We consider *quantified Boolean formulas* in *prenex conjunctive normal form* (PCNF), which are of the form $\mathcal{Q}.\psi$, where $\mathcal{Q}$ is a *quantifier prefix* (as defined in the following) and ψ, called the *matrix* of the QBF, is a propositional formula in CNF. A quantifier prefix has the form $\mathcal{Q}_1 X_1 \dots \mathcal{Q}_q X_q$, where all the X_i are mutually disjoint sets of variables, $\mathcal{Q}_i \in \{\forall, \exists\}$, and $\mathcal{Q}_i \neq \mathcal{Q}_{i+1}$. The quantifier of a literal l is $\mathcal{Q}_i$ if $var(l) \in X_i$. Given a literal l with quantifier $\mathcal{Q}_i$ and a literal k with quantifier $\mathcal{Q}_j$, we write $l \leq_{\mathcal{Q}} k$ if $i \leq j$, and $l <_{\mathcal{Q}} k$ if $i < j$. If $l \leq_{\mathcal{Q}} k$, we say that l occurs *outer* to k and that k occurs *inner* to l. Moreover, if $\mathcal{Q}$ is clear from the context, we sometimes omit it and just write $l \leq k$ or $l < k$.

Before we define the semantics of quantified Boolean formulas in PCNF, remember that for a propositional formula ψ and an assignment α, we denote by $\psi|_\alpha$ the result of first removing from ψ all clauses that are satisfied by α and then removing from the remaining clauses all literals that are falsified by α. A QBF $\exists x\mathcal{Q}.\psi$ is true if at least one of $\mathcal{Q}.\psi|_x$ and $\mathcal{Q}.\psi|_{\bar{x}}$ is true, otherwise it is false. Respectively, a QBF $\forall x\mathcal{Q}.\psi$ is true if both $\mathcal{Q}.\psi|_x$ and $\mathcal{Q}.\psi|_{\bar{x}}$ are true, otherwise it is false. If the matrix ψ of a QBF $\mathcal{Q}.\psi$ is the empty formula, then $\mathcal{Q}.\psi$ is true. If ϕ contains the empty clause, then $\mathcal{Q}.\psi$ is false.

The formal definition of QBF proof systems is analogous to that of propositional proof systems, again following Cook and Reckhow [CR79]:

Definition 53. *A* proof system *for (false) quantified Boolean formulas in PCNF is a surjective polynomial-time-computable function $f : \Sigma^* \to \mathcal{F}$ where Σ is some alphabet and $\mathcal{F}$ is the set of all false QBFs.*

Polynomial simulations of proof systems are then defined as follows [CR79]:

Definition 54. *A proof system $f_1 : \Sigma_1^* \to \mathcal{F}$* polynomially simulates *a proof system $f_2 : \Sigma_2^* \to \mathcal{F}$ if there exists a polynomial-time-computable function $g : \Sigma_2^* \to \Sigma_1^*$ such that $f_1(g(x)) = f_2(x)$.*

In other words, f_1 polynomially simulates f_2 if there exists a polynomial-time-computable function that transforms f_2-proofs into f_1-proofs.

6.1.1 Resolution-Based Proof Systems

In resolution-based proof systems, a proof P of a QBF $\mathcal{Q}.\psi = \mathcal{Q}.C_1 \wedge \cdots \wedge C_m$ is a sequence $C_{m+1}, \ldots, C_n$ of clauses where $C_n = \bot$, and where for every C_i $(m+1 \leq i \leq n)$, it holds that C_i has been derived from clauses in ψ or from earlier clauses in P (i.e., from clauses with index strictly smaller than i) by applications of either the $\forall$-*red* rule (also called *universal reduction*) or instantiations of the *resolution* rule which are defined as follows:

$$\frac{C \vee l}{C}\ (\forall\text{-red}) \qquad\qquad \frac{C \vee l \qquad D \vee \bar{l}}{C \vee D}\ (\text{resolution})$$

The rule $\forall$-red is only applicable if the literal l is universal and if for every existential literal $k \in C$, it holds that $k <_{\mathcal{Q}} l$. We assume that ψ contains no tautologies, otherwise the $\forall$-red rule is unsound. In the resolution rule, we say the resolvent $(C \vee D)$ is derived from its two *antecedent clauses* $(C \vee l)$ and $(D \vee \bar{l})$. Depending on the preconditions we define for the resolution rule, we obtain different proof systems.

The most basic resolution-based proof system for QBF is the so-called *Q-resolution calculus* (Q-Res) [KKF95]. It uses the resolution rule *Q-res* which requires that (1) l is existential and that (2) C does not contain a literal k such that $\bar{k} \in D$. In contrast, the *long-distance-resolution calculus* (LQ-Res) [ZM02, BJ12] uses a less restrictive variant of the resolution rule, called *LQ-res*, which requires that (1) l is existential and that (2) for every literal $k \in C$ such that $\bar{k} \in D$, it holds that k is universal and $l <_{\mathcal{Q}} k$. Note that every Q-res step is also an LQ-res step. In the rest of this chapter, we refer to resolution steps as LQ-res steps only if they are not Q-res steps, otherwise we refer to them as Q-res steps. Note that in the literature a complementary pair $x, \bar{x}$ is also represented by a so-called *merged literal* x^*.

Example 47. *Consider the QBF* $\phi = \exists a \forall x \exists b \exists c.(\bar{a} \vee \bar{x} \vee c) \wedge (\bar{x} \vee b \vee \bar{c}) \wedge (a \vee x \vee b) \wedge (\bar{b})$. *The following is a long-distance-resolution proof of* ϕ: $\bar{a} \vee \bar{x} \vee b$, $x \vee \bar{x} \vee b$, $x \vee \bar{x}$, x, $\perp$. *We explain this proof in more detail later (also see Figure 6.1 on page 102).*

6.1.2 The Proof System QRAT Light

To simulate long-distance resolution, we do not need the full power of the QRAT proof system [HSB16]. We therefore introduce only a restricted (simpler) version of QRAT. One of the main concepts in this variant of QRAT is the concept of a *blocked literal*, which is closely related to blocked clauses. For the definition of blocked literals, we first have to introduce so-called *outer resolvents*:

Definition 55. *Given two clauses* $(C \vee l), (D \vee \bar{l})$ *of a QBF* $\mathcal{Q}.\psi$, *the* outer resolvent $(C \vee l) \otimes_l^{\mathcal{Q}} (D \vee \bar{l})$ *of* $(C \vee l)$ *with* $(D \vee \bar{l})$ *upon* l *is the clause consisting of all literals in* C *together with those literals of* D *that occur outer to* l, *i.e., the outer resolvent is the clause* $C \cup \{k \mid k \in D \text{ and } k \leq_{\mathcal{Q}} l\}$.

Notice that the outer-resolvent operation is not commutative. We can now define blocked literals:

Definition 56. *A universal literal* l *is* blocked *in a clause* $(C \vee l)$ *with respect to a QBF* $\mathcal{Q}.\psi$ *if, for every clause* $(D \vee \bar{l}) \in \psi \setminus \{C \vee l\}$, *the outer resolvent* $(C \vee l) \otimes_l^{\mathcal{Q}} (D \vee \bar{l})$ *is a tautology.*

Example 48. *Let* $\phi = \exists a \forall x, y \exists b.(a \vee x \vee y) \wedge (\bar{a} \vee \bar{x} \vee b) \wedge (\bar{y} \vee \bar{x} \vee b)$. *The literal* x *is blocked in* $(a \vee x \vee y)$ *with respect to* ϕ: *There are two outer resolvents of* $(a \vee x \vee y)$ *upon* x, *namely* $(a \vee y \vee \bar{a})$, *obtained by resolving with* $(\bar{a} \vee \bar{x} \vee b)$, *and* $(a \vee y \vee \bar{y})$, *obtained by resolving with* $(\bar{y} \vee \bar{x} \vee b)$. *Both are tautologies.*

The removal of a literal that is blocked in a clause is called *blocked-literal elimination* (BLE) [HSB15]. If, after adding a literal to a clause, the literal is blocked in that clause, then this addition is called *blocked-literal addition* (BLA). Both BLE and BLA do not change the truth value of a formula.

In our restricted variant of QRAT, a *derivation* for a QBF ϕ is a sequence $M_1, \dots, M_n$ of proof steps. Starting with $\phi_0 = \phi$, every M_i modifies ϕ_{i-1} in one of the following four ways, which results in a new formula ϕ_i:

(1) It adds to ϕ_{i-1} a clause that is derived from two clauses in ϕ_{i-1} via an (unrestricted) resolution step.

(2) It adds to ϕ_{i-1} a clause C that is obtained from a clause $(C \lor l) \in \phi_{i-1}$ by a $\forall$-red step, with the additional restriction that C does not contain $\bar{l}$.

(3) It adds a blocked literal to a clause in ϕ_{i-1}.

(4) It removes a blocked literal from a clause in ϕ_{i-1}.

A QRAT derivation $M_1, \ldots, M_n$ therefore gradually derives new formulas $\phi_1, \ldots, \phi_n$ from the starting formula ϕ. If the final formula ϕ_n contains the empty clause $\bot$, then the derivation is a (*refutation*) *proof* of ϕ. Note that the $\forall$-red rule in QRAT is more restricted than the $\forall$-red rule from the resolution-based proof systems, making it sound also when clauses contain complementary literals.

To simplify the presentation, we do not specify how the modification steps M_i are represented syntactically. We also do not include clause deletion. Note that certain proof steps can modify the quantifier prefix by introducing new or removing existing variables. Note also that Q-resolution proofs do not contain complementary literals, so they can be simply rewritten into QRAT proofs using only Q-res and $\forall$-red steps. Finally, we want to highlight that for our simulation, we do not need the unrestricted resolution rule; the Q-res rule suffices.

6.2 Illustration of the Simulation

We illustrate by an example how our restricted variant of QRAT can simulate the long-distance-resolution calculus. As already mentioned, the $\forall$-red rule used in QRAT is more restricted than the one in the long-distance-resolution calculus because it does not allow us to remove a literal l from a clause that contains $\bar{l}$. This means that once we derive a clause that contains both a literal l and its complement $\bar{l}$, we cannot simply get rid of the two literals by using the $\forall$-red rule. We therefore want to avoid the derivation of clauses with complementary literals entirely. Now, the only way the long-distance-resolution calculus can derive such clauses is via resolution (LQ-res) steps. So to avoid the complementary literals, we eliminate them already before performing the resolution steps. We demonstrate this on an example:

Example 49. *Consider the QBF $\phi = \exists a \forall x \exists b \exists c.(\bar{a} \lor \bar{x} \lor c) \land (\bar{x} \lor b \lor \bar{c}) \land (a \lor x \lor b) \land (\bar{b})$ from Example 47. To increase readability, we illustrate its long-distance-resolution proof as a proof tree in Figure 6.1. To simulate this proof with* QRAT*, we first add the resolvent $(\bar{a} \lor \bar{x} \lor b)$ to ϕ via a Q-res step to obtain the new formula ϕ'. Now we cannot simply perform the next derivation step (the LQ-res step) because the resulting resolvent $(x \lor \bar{x} \lor b)$ would contain complementary literals. To deal with this, we try to eliminate x from the clause $(a \lor x \lor b)$. This is where the addition and elimination of blocked literals come into play.*

$$\dfrac{\dfrac{\dfrac{a \lor x \lor b \qquad \dfrac{\bar{x} \lor b \lor \bar{c} \qquad \bar{a} \lor \bar{x} \lor c}{\bar{a} \lor \bar{x} \lor b}\ (\text{Q-res})}{x \lor \bar{x} \lor b}\ (\text{LQ-res}) \qquad \bar{b}}{x \lor \bar{x}}\ (\text{Q-res})}{\dfrac{x}{\bot}\ (\forall\text{-red})}\ (\forall\text{-red})$$

Figure 6.1: LQ-res proof of QBF $\phi = \exists a \forall x \exists b \exists c.(\bar{a} \lor \bar{x} \lor c) \land (\bar{x} \lor b \lor \bar{c}) \land (a \lor x \lor b) \land (\bar{b})$.

We cannot yet eliminate x from ϕ' because x is not blocked in $(a \lor x \lor b)$ with respect to ϕ': For x to be blocked, all outer resolvents of $(a \lor x \lor b)$ upon x must contain complementary literals. The clauses that can be resolved with $(a \lor x \lor b)$ are $(\bar{a} \lor \bar{x} \lor c)$, $(\bar{a} \lor \bar{x} \lor b)$, and $(\bar{x} \lor b \lor \bar{c})$. While the outer resolvents with the former two clauses contain the complementary literals a and $\bar{a}$, the outer resolvent $(a \lor b)$, obtained by resolving with $(\bar{x} \lor b \lor \bar{c})$, does not contain complementary literals.

Now we use a feature of QRAT *to make x blocked in $(a \lor x \lor b)$: We add a new literal x' (which goes to the same quantifier block as x) to $(a \lor x \lor b)$ to turn it into $(a \lor x' \lor x \lor b)$. The addition of x' is clearly a blocked-literal addition as there are no outer resolvents of $(a \lor x' \lor x \lor b)$ upon x'. Likewise, we add the complement $\bar{x}'$ of x' to $(\bar{x} \lor b \lor \bar{c})$ to turn it into $(\bar{x}' \lor \bar{x} \lor b \lor \bar{c})$. Again this is a blocked-literal addition since $(a \lor x' \lor x \lor b)$ (which is the only clause containing the complement x' of $\bar{x}'$) contains x while $(\bar{x}' \lor \bar{x} \lor b \lor \bar{c})$ contains $\bar{x}$.*

Now the complementary pair $x', \bar{x}'$ is contained in the outer resolvent of $(a \lor x' \lor x \lor b)$ with $(\bar{x}' \lor \bar{x} \lor b \lor \bar{c})$ upon x. Thus, the literal x becomes blocked in $(a \lor x' \lor x \lor b)$ and so we can remove it to obtain $(a \lor x' \lor b)$. We have thus replaced x in $(a \lor x \lor b)$ by x' and now we can resolve $(a \lor x' \lor b)$ with $(\bar{a} \lor \bar{x} \lor b)$ upon a to obtain the resolvent $(x' \lor \bar{x} \lor b)$ (instead of $(x \lor \bar{x} \lor b)$ as in the original proof). Finally, we resolve $(x' \lor \bar{x} \lor b)$ with $\bar{b}$ to obtain $(x' \lor \bar{x})$ from which we derive the empty clause $\bot$ via $\forall$-red steps.

To summarize, we start by simply adding clauses of a given long-distance-resolution proof to our formula until we encounter an LQ-res step. To avoid complementary literals in the resolvent of the LQ-res step, we then use blocked-literal addition and blocked-literal elimination to replace these literals. After this, we can derive a resolvent without complementary literals and move on until we encounter the next LQ-res step, which we again eliminate. We repeat this procedure until the whole long-distance-resolution proof is turned into a QRAT proof.

Note that the modification of existing clauses has an impact on later derivations. For instance, by replacing $(a \lor x \lor b)$ in the above example with $(a \lor x' \lor b)$, we not only affected the immediate resolvent $(x \lor \bar{x} \lor b)$, which we turned into $(x' \lor \bar{x} \lor b)$, but also the later resolvent $(x \lor \bar{x})$, which became $(x' \lor \bar{x})$. We therefore have to show that these modifications are harmless in the sense that they do not lead to an invalid proof. We do so in the next section, where we define our simulation in detail before proving that it indeed produces a valid QRAT proof.

6.3 Simulation

We first describe our simulation procedure on a high level before we specify the details and prove its correctness. As we have seen, given a long-distance-resolution proof, we can use QRAT to derive all clauses up to the first LQ-res step. The crucial part of the simulation is then the elimination of complementary literals from this LQ-res step, which might involve the modification of several clauses via the addition and elimination of blocked literals.

Let $\phi = \mathcal{Q}.C_1 \wedge \cdots \wedge C_m$ be a QBF and let $P = C_{m+1}, \ldots, C_r, \ldots, C_n$ be a long-distance-resolution proof of ϕ where C_r is the first clause derived via an LQ-res step. If there is no such C_r, the proof can be directly translated to QRAT. Otherwise, in a first step, our procedure produces a QRAT derivation that adds all the clauses $C_{m+1}, \ldots, C_{r-1}$ to ϕ by using Q-res and $\forall$-red steps. It then uses blocked-literal addition and blocked-literal elimination to avoid complementary literals in the resolvent C_r, which it thereby turns into a different resolvent C_r'. After this, it adds C_r' to ϕ via a Q-res step. The result is a QRAT derivation of a formula ϕ' from ϕ. We explain this first step in Section 6.3.1.

In a second step, the procedure first removes all the clauses $C_{m+1}, \ldots, C_r$ from P since they—or their modified variants—are now all contained in ϕ'. As several clauses have been modified via blocked-literal addition and blocked-literal elimination in the first step, it then propagates these modifications through the remaining part of P. This turns P into a long-distance resolution proof P' of ϕ'. We explain this second step in Section 6.3.2.

By repeating these two steps for every LQ-res step, we finally obtain a QRAT proof of ϕ. Thus, we have to show that after the above two steps (i.e., after one iteration of our procedure), ϕ' is obtained by a valid QRAT derivation and the proof P' is a valid long-distance-resolution proof of ϕ' that is shorter than P. The correctness of the simulation follows then simply by induction.

To simplify the presentation, we assume that the long-distance resolvent C_r contains only one pair of complementary literals, i.e., $C_r = (C \vee D \vee x \vee \bar{x})$ was derived from two clauses $(C \vee l \vee x)$ and $(D \vee \bar{l} \vee \bar{x})$ where C does not contain a literal k such that $\bar{k}$ is contained in D. Although this assumption leads to a loss of generality, we show later that our argument can be easily extended to the more general case where C and D are allowed to contain multiple pairs of complementary literals.

6.3.1 QRAT Derivation of the Formula ϕ'

Below we describe the QRAT derivation of ϕ' from ϕ. Initially, $\phi' = \phi$.

1. Add the clauses $C_{m+1}, \ldots C_{r-1}$ to ϕ' via Q-res and $\forall$-red steps.

2. Consider the LQ-res step that derived $C_r = (C \lor D \lor x \lor \bar{x})$ from two clauses $(C \lor l \lor x)$ and $(D \lor \bar{l} \lor \bar{x})$:

$$\frac{C \lor l \lor x \qquad D \lor \bar{l} \lor \bar{x}}{C \lor D \lor x \lor \bar{x}}\ \text{(LQ-res)}$$

   Towards making x blocked in $(C \lor l \lor x)$, add a new literal x' (that goes to the same quantifier block as x) to $(C \lor l \lor x)$ to turn it into $(C \lor l \lor x' \lor x)$.

3. Add $\bar{x}'$ to each clause $C_i \in \phi'$ for which (1) $\bar{x} \in C_i$, and (2) the outer resolvent of $(C \lor l \lor x' \lor x)$ and C_i upon x is not a tautology.

4. Now x is a blocked literal in $(C \lor l \lor x' \lor x)$. Eliminate it to obtain $(C \lor l \lor x')$.

5. Add the clause $(C \lor D \lor x' \lor \bar{x})$ to ϕ' by performing a Q-res step of $(C \lor l \lor x')$ and $(D \lor \bar{l} \lor \bar{x})$ upon l.

To see that this results in a valid QRAT derivation, observe the following: In step 2, the addition of x' is a blocked-literal addition, since $\bar{x}'$ is not contained in any of the clauses. In step 3, for every C_i with $\bar{x} \in C_i$, the addition of $\bar{x}'$ is a blocked-literal addition as only $(C \lor l \lor x' \lor x)$ can be resolved with C_i upon $\bar{x}'$ and the corresponding outer resolvent contains x and $\bar{x}$. Note that instead of eliminating x from $(C \lor l \lor x)$, we could have also eliminated $\bar{x}$ from $(D \lor \bar{l} \lor \bar{x})$. It remains to modify the long-distance-resolution proof P of ϕ so that it becomes a valid proof of ϕ'.

6.3.2 Modification of the Long-Distance-Resolution Proof

We turn the proof $P = C_{m+1}, \ldots, C_r, \ldots, C_n$ of ϕ into a proof P' of ϕ'. First, we remove the clauses $C_{m+1}, \ldots, C_r$ from P since ϕ' already contains variants $C'_{m+1}, \ldots, C'_r$ of these clauses. Second, since we have modified the clauses in ϕ', we have to propagate these modifications through the remaining proof.

Assume, for instance, that in P the clause C_{r+1} has been obtained by resolving a clause C_i with a clause C_j. Both C_i and C_j might have been affected by blocked-literal additions so that they are now different clauses $C'_i, C'_j \in \phi'$. To account for these modifications of C_i and C_j, we replace C_{r+1} in P by the resolvent of C'_i and C'_j. Moreover, in cases where P removes x from a clause via a $\forall$-red step, we now also remove x'. Analogously for $\bar{x}'$ and $\bar{x}$.

To formalize these modifications, we first assign to every clause C_i with $1 \leq i \leq r$ its corresponding clause of ϕ' as follows:

$$C'_i = \begin{cases} C_i \cup \{\bar{x}'\} & \text{if } \bar{x} \in C_i \text{ and the outer resolvent of } (C \lor l \lor x \lor x') \\ & \text{and } C_i \text{ upon } x \text{ is not a tautology;} \\ (C_i \setminus \{x\}) \cup \{x'\} & \text{if } C_i = C_r \text{ or } C_i = (C \lor l \lor x); \\ C_i & \text{otherwise.} \end{cases}$$

Note that, by construction, $C'_i \in \phi'$ for $1 \leq i \leq r$. For every i such that $r < i \leq n$, we step-by-step, starting with $i = r + 1$, define C'_i based on the derivation rule that was used for deriving C_i in P. We distinguish between clauses derived by resolution steps and clauses derived by ∀-red steps:

(1) C_i has been derived via a resolution step of two clauses $C_j = (C \vee l)$ and $C_k = (D \vee \bar{l})$ upon l, i.e., $C_i = (C \vee D)$. We define $C'_i = C'_j \setminus \{l\} \vee C'_k \setminus \{\bar{l}\}$.

(2) C_i has been derived from a clause C_j via a ∀-red step. If the ∀-red step removes a literal l with $var(l) \neq var(x)$, we define $C'_i = C'_j \setminus \{l\}$. If it removes x, we define $C'_i = C'_j \setminus \{x, x'\}$, and if it removes $\bar{x}$, we define $C'_i = C'_j \setminus \{\bar{x}, \bar{x}'\}$.

Note that ∀-red steps of x and $\bar{x}$ in P' might remove two literals at once. Although such ∀-red steps do not constitute valid derivation steps in a strict sense, this is not a serious problem. These steps can be easily rewritten into two distinct ∀-red steps since x and x' are in the same quantifier block. For instance, the left step below can be rewritten into the two steps on the right:

$$\frac{C \vee x \vee x'}{C}\ (\forall\text{-red}) \qquad\qquad \cfrac{\cfrac{C \vee x \vee x'}{C \vee x}\ (\forall\text{-red})}{C}\ (\forall\text{-red})$$

Next, we show that the resulting proof P' is—apart from the minor detail just mentioned—a valid long-distance-resolution proof of ϕ'.

6.4 Correctness of the Simulation

To prove the correctness of our simulation, we first introduce a lemma that guarantees that the modified long-distance-resolution proof P' has a similar structure as the original proof P:

Lemma 56. *Let $\phi' = \mathcal{Q}'.C'_1 \wedge \cdots \wedge C'_r$ and $P' = C'_{r+1}, \ldots, C'_n$ be obtained from $\phi = \mathcal{Q}.C_1 \wedge \cdots \wedge C_m$ and $P = C_{m+1}, \ldots, C_r, \ldots, C_n$ as defined above. Then, for every clause C'_i with $1 \leq i \leq n$, the following holds: (1) If x' or x is in C'_i, then $x \in C_i$. (2) If $\bar{x}'$ or $\bar{x}$ is in C'_i, then $\bar{x} \in C_i$. (3) C'_i agrees with C_i on all literals whose variables are different from x and x', i.e., $C'_i \setminus \{x, \bar{x}, x', \bar{x}'\} = C_i \setminus \{x, \bar{x}\}$.*

Proof. By induction on i.

Base Case ($i \leq r$): The claim holds by the definition of C'_i.

Induction Step ($r < i$): Consider the clause C_i in P that corresponds to C'_i. We proceed by a case distinction based on how C_i was derived in P.

Case 1: C_i is a resolvent $C_j \setminus \{l\} \vee C_k \setminus \{\bar{l}\}$ of two clauses C_j, C_k. In this case, $C'_i = C'_j \setminus \{l\} \vee C'_k \setminus \{\bar{l}\}$. By the induction hypothesis, the statement holds for C'_j and C'_k. Now, if C'_i contains x' or x, then at least one of C'_j and C'_k must contain x' or x and thus one of C_j and C_k must contain x, hence $x \in C_i$. Analogously, if C'_i contains $\bar{x}'$ or $\bar{x}$, then C_i contains $\bar{x}$. Now, C'_j agrees with C_j on all literals whose variables are different from x and x', and the same holds for C'_k and C_k. Thus, C'_i agrees with C_i on all literals whose variables are different from x and x'.

Case 2: C_i has been derived from a clause C_j via a $\forall$-red step, i.e., $C_i = C_j \setminus \{l\}$ for some universal literal l. By the induction hypothesis, the statement holds for C'_j. If $var(l) \neq var(x')$, then $C'_i = C'_j \setminus \{l\}$ and thus the claim holds. If $l = x$, then $C'_i = C'_j \setminus \{x, x'\}$ and thus the claim holds too. The case where $l = \bar{x}$ is analogous to the case where $l = x$. □

We can now show that the proof P', produced by our simulation procedure, is a valid long-distance-resolution proof of ϕ':

Theorem 57. *Let $\phi' = \mathcal{Q}'.C'_1 \wedge \cdots \wedge C'_r$ and $P' = C'_{r+1}, \ldots, C'_n$ be obtained from $\phi = \mathcal{Q}.C_1 \wedge \cdots \wedge C_m$ and $P = C_{m+1}, \ldots, C_r, \ldots, C_n$ by our procedure. Then, P' is a valid long-distance-resolution proof of ϕ'.*

Proof. We have to show that every clause C'_i in P' has been derived from clauses in $C'_1, \ldots, C'_{i-1}$ via a valid application of a derivation rule and that $C'_n = \bot$. To show that every clause in P' has been derived via a valid application of a derivation rule, let C'_i be a clause in P'. We proceed by a case distinction based on the rule via which its counterpart C_i has been derived in P:

Case 1: C_i has been derived from two clauses C_j, C_k via a Q-res step or an LQ-res step upon some existential literal l. In this case, $C'_i = C'_j \setminus \{l\} \vee C'_k \setminus \{\bar{l}\}$. We have to show that $l \in C'_j$, $\bar{l} \in C'_k$, and that for every literal $l' \in C'_j$ such that $l' \neq l$ and $\bar{l}' \in C'_k$, it holds that l' is universal and $l <_{\mathcal{Q}'} l'$. By Lemma 56, C'_j agrees with C_j on all literals whose variables are different from the universal literals x and x'. Likewise for C'_k and C_k. Therefore, $l \in C'_j$ and $\bar{l} \in C'_k$.

Now, assume C'_j contains a literal l' such that $l' \neq l$ and $\bar{l}' \in C'_k$. If the variable of l' is different from x and x', then it must be the case that l' is universal and $l <_{\mathcal{Q}'} l'$, for otherwise the derivation of C_i in P were not valid. Assume thus that the variable of l' is either x or x'. If l' is either x or x', then Lemma 56 implies that C_j contains x and also, since $\bar{l}' \in C'_k$, that C_k contains $\bar{x}$. Therefore, it holds that $l <_{\mathcal{Q}'} x$ (since otherwise the derivation of C_i in P were not valid) and since x' and x are in the same quantifier block, it also holds that $l <_{\mathcal{Q}'} x'$, hence $l <_{\mathcal{Q}'} l'$. The case where l' is $\bar{x}$ or $\bar{x}'$ is symmetric.

Case 2: C_i has been derived from a clause C_j via a $\forall$-red step, that is, by removing a universal literal l such that for every existential literal $l' \in C_j$, it holds that $l' <_{\mathcal{Q}} l$.

If $var(l) \neq x$, then $C'_j = C'_i \setminus \{l\}$ and since, by Lemma 56, C'_i coincides with C_i on all existential variables, it holds for every existential literal $l' \in C'_i$ that $l' <_{\mathcal{Q}'} l$. If $var(l) = x$, then C'_j is of the form $C'_i \setminus \{x, x'\}$ or $C'_i \setminus \{\bar{x}, \bar{x}'\}$. Now, x and x' are in the same quantifier block and thus, with the same argument as for $var(l) = x$, it holds for every existential literal $l' \in C'_j$ that $l' <_{\mathcal{Q}'} l$.

Finally, to see that $C'_n = \bot$, observe the following: By Lemma 56, since x and $\bar{x}$ are not in C_n, it follows that x' and $\bar{x}'$ are not in C'_n. Moreover, again by Lemma 56, C_n and C'_n agree on all other literals. Therefore, $C'_n = C_n = \bot$. ☐

We can also show that our simulation does not introduce new LQ-res steps. Hence, if a long-distance-resolution proof contains n LQ-res steps, our simulation terminates after at most n iterations (the proof is omitted due to space reasons):

Theorem 58. *Let P' be obtained from $\phi = \mathcal{Q}.\psi$ and P by our procedure. Then, P' contains fewer LQ-res steps than P.*

Until now, we have assumed that LQ-res steps involve only a single pair of complementary universal literals. When multiple such pairs are involved, the procedure changes only slightly: Instead of eliminating only a single literal from one of the clauses involved in the LQ-res step, we now eliminate several of them. If we start with the outermost such literal and gradually move inwards, we ensure that at most one blocked literal is added per clause. As an example, consider the following derivation in a long-distance-resolution proof of the QBF $\phi = \exists a \exists b \forall x \exists c \forall y \exists d.(b \vee x \vee c \vee y \vee d) \wedge (a \vee \bar{x} \vee c) \wedge (\bar{a} \vee \bar{b} \vee \bar{y} \vee d)$:

$$\dfrac{b \vee x \vee c \vee y \vee d \qquad \dfrac{a \vee \bar{x} \vee c \qquad \bar{a} \vee \bar{b} \vee \bar{y} \vee d}{\bar{b} \vee \bar{x} \vee c \vee \bar{y} \vee d}\,\text{(Q-res)}}{x \vee \bar{x} \vee c \vee y \vee \bar{y} \vee d}\,\text{(LQ-res)}$$

In the LQ-res step, there are two pairs of complementary universal literals, namely $x, \bar{x}$ and $y, \bar{y}$. We therefore try to get rid of both x and y in the left antecedent $L = (b \vee x \vee c \vee y \vee d)$ of the LQ-res step. As in the case where only one literal is removed, we start by deriving in QRAT all clauses that occur before the LQ-res step. In this case, we add $(\bar{b} \vee \bar{x} \vee c \vee \bar{y} \vee d)$ to ϕ via a Q-res step and denote the resulting formula by ϕ'.

Now we want to remove x from L via blocked-literal elimination. In order for x to be blocked in ϕ', all outer resolvents of L upon x have to be tautologies. The formula ϕ' contains two clauses that can be resolved with L upon x, namely $(\bar{b} \vee \bar{x} \vee c \vee \bar{y} \vee d)$ and $(a \vee \bar{x} \vee c)$. As the first clause contains $\bar{b}$ and L contains b, the corresponding outer resolvent upon x contains $b, \bar{b}$. But there are no complementary literals in the outer resolvent $(a \vee b)$ with the second clause. We therefore add a fresh literal x' to L and add its complement $\bar{x}'$ to $(\bar{a} \vee \bar{x} \vee c)$ to obtain $\phi' = \exists a \exists b \forall x \forall x' \exists c \forall y \exists d.(b \vee x \vee x' \vee c \vee y \vee d) \wedge (a \vee \bar{x} \vee \bar{x}' \vee c) \wedge (\bar{a} \vee \bar{b} \vee \bar{y} \vee d) \wedge (\bar{b} \vee \bar{x} \vee c \vee \bar{y} \vee d)$.

We can now remove the blocked literal x from $(b \vee x \vee x' \vee c \vee y \vee d)$ to obtain the clause $L' = (b \vee x' \vee c \vee y \vee d)$. If we now resolved L' with $(\bar{b} \vee \bar{x} \vee c \vee \bar{y} \vee d)$, we would get the following LQ-res step:

$$\frac{b \vee x' \vee c \vee y \vee d \qquad \bar{b} \vee \bar{x} \vee c \vee \bar{y} \vee d}{x' \vee \bar{x} \vee c \vee y \vee \bar{y} \vee d} \text{ (LQ-res)}$$

Since there is still a clash of y and $\bar{y}$, we need to get rid of y in L'. We can do this without performing any blocked-literal additions: The only clauses in ϕ' that contain $\bar{y}$ are $(\bar{a} \vee \bar{b} \vee \bar{y} \vee d)$ and $(\bar{b} \vee \bar{x} \vee c \vee \bar{y} \vee d)$, and the outer resolvents of L' with both of them contain complementary literals. We can thus remove y from L' and use a Q-res step to add the resulting resolvent to ϕ':

$$\frac{b \vee x' \vee c \vee d \qquad \bar{b} \vee \bar{x} \vee c \vee \bar{y} \vee d}{x' \vee \bar{x} \vee c \vee \bar{y} \vee d} \text{ (Q-res)}$$

Similar to the case where we only eliminated one literal, we then propagate the corresponding changes through the rest of the proof to turn it into a valid long-distance resolution proof of ϕ'.

6.5 Complexity of the Simulation

After showing how a long-distance-resolution proof can be translated into a QRAT proof, we still have to prove that the size (the number of derivation steps) of the resulting QRAT proof is polynomial with respect to the size of the original proof and the formula. We have seen that the long-distance-resolution proof and the QRAT proof correspond one-to-one on resolution steps and $\forall$-red steps. Therefore, we only need to estimate the number of blocked-literal addition and blocked-literal elimination steps to obtain an upper bound on the size of the QRAT proof.

Consider a long-distance-resolution proof $C_{m+1}, \dots, C_r, \dots, C_n$ of a QBF $\mathcal{Q}.C_1 \wedge \cdots \wedge C_m$, where C_r is the first clause that is derived via an LQ-res step:

$$\frac{C \vee l \vee k_1 \vee \cdots \vee k_p \qquad D \vee \bar{l} \vee \bar{k}_1 \vee \cdots \vee \bar{k}_p}{C_r = C \vee D \vee k_1 \vee \bar{k}_1 \vee \cdots \vee k_p \vee \bar{k}_p} \text{ (LQ-res)}$$

We can make the following observation: To remove all the universal literals $k_1, \dots, k_p$ from $(C \vee l \vee k_1 \vee \cdots \vee k_p)$ via blocked-literal elimination, we have to add at most one new literal of the form $\bar{k}_i'$ to every clause $C_1, \dots, C_{r-1}$ if we start by eliminating the outermost universal literal k_1 and step-by-step work ourselves towards the innermost literal k_p. The reason this works is as follows:

Assume we have added the literal k_1' to $(C \vee l \vee k_1 \vee \cdots \vee k_p)$ and the corresponding literal $\bar{k}_1'$ to another clause $C_i = (C_i' \vee \bar{k}_1)$ to obtain complementary literals in the outer resolvent of the resulting clauses $(C \vee l \vee k_1 \vee k_1' \vee \cdots \vee k_p)$ and $(C' \vee \bar{k}_1 \vee \bar{k}_1')$ upon k_1. Then, the outer resolvent of $(C \vee l \vee k_1 \vee k_1' \vee \cdots \vee k_p)$ with $(C' \vee \bar{k}_1 \vee \bar{k}_1')$ upon a literal k_j that is inner to k_1 (i.e., $k_1 <_{\mathcal{Q}} k_j$) contains the complementary pair $k_1', \bar{k}_1'$, so we have to add no further literals to $(C' \vee \bar{k}_1 \vee \bar{k}_1')$.

Hence, the number of blocked-literal additions for literals of the form $\bar{k}_i'$ is bounded by the number of clauses, that is, by n. Moreover, for every addition of a literal $\bar{k}_i'$ to some clause, there is at most one addition of the corresponding literal k_i'. Therefore, there are at most $2n$ blocked-literal additions per LQ-res step. Now, for every addition of a literal k_i', there is exactly one elimination of the corresponding literal k_i. Thus, overall there are at most $3n$ blocked-literal additions and eliminations for every LQ-res step. Since the number of LQ-res steps is bounded by the number of clauses in the proof, the size of the QRAT derivation is at most $3n^2$. It follows that whenever a QBF has a long-distance-resolution proof of polynomial size, it also has a polynomial-size QRAT proof. As all the steps in the transformation are straightforward, it should then be clear that this transformation can be performed in polynomial time. We thus get:

Theorem 59. *The* QRAT *proof system polynomially simulates the long-distance-resolution calculus.*

6.6 Empirical Evaluation

We now know that QRAT can polynomially simulate long-distance resolution. But what does it mean in practice? Can we have short QRAT proofs for formulas that have short long-distance-resolution proofs? To answer this question at least partly, we consider the formulas well-known for having short long-distance-resolution proofs while only having long Q-resolution proofs—the Kleine Büning formulas [KKF95]. A Kleine Büning formula of size n, in short KBKF_n, has the prefix $\exists a_0, a_1, b_1 \forall x_1 \exists a_2, b_2 \forall x_2 \ldots \exists a_n, b_n \forall x_n \exists c_1, c_2, \ldots, c_n$ and the following clauses:

$$
\begin{array}{llll}
I \;:\; \bar{a}_0 & I' \;:\; a_0 \vee \bar{a}_1 \vee \bar{b}_1 & \\
A_i :\; a_i \vee \bar{x}_i \vee \bar{a}_{i+1} \vee \bar{b}_{i+1} & B_i :\; b_i \vee x_i \vee \bar{a}_{i+1} \vee \bar{b}_{i+1} & \text{for } i \in \{1..n-1\} \\
C \;:\; a_n \vee \bar{x}_n \vee \bar{c}_1 \vee \cdots \vee \bar{c}_n & C' :\; b_n \vee x_n \vee \bar{c}_1 \vee \cdots \vee \bar{c}_n & \\
X_i :\; \bar{x}_i \vee c_i & X_i' :\; x_i \vee c_i & \text{for } i \in \{1..n\}
\end{array}
$$

We can reduce a formula KBKF_n to a formula KBKF_{n-1} by using only Q-res, blocked-literal elimination, and clause-deletion steps[1] (no $\forall$-red steps or resolution upon universal literals). To do so, we use the clauses A_n, B_n, C, C', X_n, and X_n' of KBKF_n to construct the clauses C and C' of KBKF_{n-1}. The required 12 steps are shown below. The last two clauses (11 and 12) respectively correspond to the clauses C and C' of KBKF_{n-1}.

[1] Clause deletion was not used in the simulation, but is allowed in the QRAT system.

Table 6.1: The size of Kleine Büning formulas in the number of variables and clauses. Additionally, the size of their long-distance-resolution proofs (in the QRP format) in the number of Q-res steps (Q), LQ-res steps (LQ), ∀-red steps (∀), and the file size in KB (ignoring the part that represents the formula). On the right, the number of Q-res (Q), BLE (BLE), and deletion (D) steps as well as the file size for the manual QRAT proofs.

	Input Formula		LD proofs (QRP)				QRAT proofs			
Formula	Variables	Clauses	Q	LQ	∀	File Size	Q	BLE	D	File Size
$KBKF_{10}$	41	42	41	18	38	6	57	38	92	6
$KBKF_{50}$	201	202	201	98	198	138	297	198	492	112
$KBKF_{100}$	401	402	401	198	398	573	597	398	992	421
$KBKF_{200}$	801	802	801	398	798	2321	1197	798	1992	1627
$KBKF_{500}$	2001	2002	2001	998	1998	16 259	2997	1998	4992	11 890

1.	$a_n \vee \bar{x}_n \vee \bar{c}_1 \vee \cdots \vee \bar{c}_{n-1}$	(Q-res of C and X_n)
2.	$b_n \vee x_n \vee \bar{c}_1 \vee \cdots \vee \bar{c}_{n-1}$	(Q-res of C' and X'_n)
3.		(delete C, C', X_n, X'_n)
4.	$a_{n-1} \vee \bar{x}_{n-1} \vee \bar{b}_n \vee \bar{x}_n \vee \bar{c}_1 \vee \cdots \vee \bar{c}_{n-1}$	(Q-res of 1 and A_{n-1})
5.	$b_{n-1} \vee x_{n-1} \vee \bar{a}_n \vee x_n \vee \bar{c}_1 \vee \cdots \vee \bar{c}_{n-1}$	(Q-res of 2 and B_{n-1})
6.	$a_{n-1} \vee \bar{x}_{n-1} \vee \bar{b}_n \vee \bar{c}_1 \vee \cdots \vee \bar{c}_{n-1}$	(BLE of $\bar{x}_n$ from 4)
7.	$b_{n-1} \vee x_{n-1} \vee \bar{a}_n \vee \bar{c}_1 \vee \cdots \vee \bar{c}_{n-1}$	(BLE of x_n from 5)
8.	$a_{n-1} \vee \bar{x}_{n-1} \vee x_n \vee \bar{c}_1 \vee \cdots \vee \bar{c}_{n-1}$	(Q-res of 6 and B_{n-1})
9.	$b_{n-1} \vee x_{n-1} \vee \bar{x}_n \vee \bar{c}_1 \vee \cdots \vee \bar{c}_{n-1}$	(Q-res of 7 and A_{n-1})
10.		(delete 4, 5, 6, 7, A_{n-1}, B_{n-1})
11.	$a_{n-1} \vee \bar{x}_{n-1} \vee \bar{c}_1 \vee \cdots \vee \bar{c}_{n-1}$	(BLE of x_n from 8)
12.	$b_{n-1} \vee x_{n-1} \vee \bar{c}_1 \vee \cdots \vee \bar{c}_{n-1}$	(BLE of $\bar{x}_n$ from 9)

Table 6.1 shows the sizes of the Kleine Büning formulas as well as of the corresponding long-distance-resolution proofs (in the QRP format) and QRAT proofs. The latter are obtained by the construction mentioned in this section. The size of both types of proofs is linear in the size of the formula. Although QRAT proofs use about twice as many proof steps (including deletion steps), the file size of QRAT proofs is smaller. The explanation for this is that long-distance-resolution proofs increase the length of clauses, while QRAT proofs decreases their length.

Short proofs of the *KBKF* formulas can also be obtained by using resolution upon universal variables as in the calculus QU-Res [VG12a]. There is, however, a variant of the *KBKF* formulas, called $KBKF_n{-}qu$ [BWJ14], which has only exponential proofs in the QU-Res calculus. A $KBKF_n{-}qu$ formula is obtained from $KBKF_n$ by adding a universal literal y_i (occurring in the same quantifier block as x_i) to every clause in $KBKF_n$ that contains x_i, and a literal $\bar{y}_i$ to every clause in $KBKF_n$. For these formulas, blocked-literal elimination can remove all the y_i and $\bar{y}_i$ literals, which reduces a $KBKF_n{-}qu$ formula to a $KBKF_n$ formula that can then be efficiently proved using resolution upon universal

Table 6.2: Comparison of the QRAT proofs obtained by applying `ld2qrat` to long-distance-resolution proofs (in the QRP format) of the Kleine Büning formulas. The file size is given in KB and the time for translating the proof (time) is given in seconds.

	QRP to QRAT Without Deletion				QRP to QRAT With Deletion			
Formula	Variables	Steps	File Size	Time	Variables	Steps	File Size	Time
$KBKF_{10}$	59	1690	103	0.07	59	448	26	0.01
$KBKF_{50}$	299	52 170	18 774	0.45	299	6288	2227	0.12
$KBKF_{100}$	599	214 270	154 299	3.77	599	22 588	16 192	0.86
$KBKF_{200}$	1199	868 470	1 309 559	30.70	1199	85 188	126 375	7.95
$KBKF_{500}$	2999	5 471 070	23 622 369	497.32	2999	512 988	2 229 195	124.10

literals.

In addition to the handcrafted QRAT proofs, we implemented a tool (called `ld2qrat`) that, based on our simulation, transforms long-distance-resolution proofs into QRAT proofs. We used `ld2qrat` to transform the long-distance-resolution proofs of the $KBKF_n$ formulas (by Egly et al. [ELW13]) into QRAT proofs and validated the correctness of these proofs with the proof checker `QRAT-trim`. In the plain mode, `ld2qrat` closely follows our simulation. Additionally, it features two optimizations: (1) Given an LQ-res step upon l with the antecedents $(C \lor l \lor x)$ and $(D \lor \bar{l} \lor \bar{x})$, if one of x or $\bar{x}$ is already a blocked literal, it is removed with blocked-literal elimination. This avoids the introduction of new variables. (2) Clauses are deleted as soon as they are not needed later in the proof anymore.

Table 6.2 shows properties of the QRAT proofs produced by `ld2qrat` from the long-distance-resolution proofs of the *KBKF* formulas. On the left are the sizes of proofs obtained without the clause-deletion optimization. On the right are the sizes of proofs with this optimization. A (least squares) regression analysis confirms that the length (number of steps) of the QRAT proofs without deletion is quadratically related to the length of the corresponding long-distance-resolution proofs: The function $f(x) = 0.22x^2 - 4.48x + 54.58$ (where x is the length of the long-distance-resolution proof and $f(x)$ is the length of the QRAT proof) fits the data from the above tables perfectly (the error term R^2 of the regression is 1).

6.7 QRAT and Other Resolution-Based Proof Systems

After the analysis of QRAT in theory and practice, we now relate it to popular resolution-based proof systems for QBF. An overview of the relationships between these systems is illustrated in Figure 6.2. Besides the long-distance-resolution calculus LQ-Res, another well-known proof system is the calculus QU-Res [VG12a], which extends the basic Q-resolution calculus (Q-Res) by allowing resolution upon universals literals if the resulting resolvent does not contain complementary literals. As QRAT also allows resolution upon universal literals, it simulates QU-Res. Balabanov et al. [BWJ14] showed the incomparability between LQ-Res and QU-Res by exponential separations. It thus follows that QRAT is strictly stronger than both LQ-Res and QU-Res.

Another proof system that is stronger than both LQ-Res and QU-Res is the calculus LQU$^+$-Res [BWJ14], which extends LQ-Res by allowing (long-distance) resolution upon universals literals. We know that either QRAT is strictly stronger than LQU$^+$-Res or the two systems are incomparable: On purely existentially-quantified formulas, LQU$^+$-Res boils down to ordinary propositional resolution (without complementary literals in resolvents) whereas the QRAT system boils down to the RAT system [WHHJ14]. As the RAT system is strictly stronger than resolution—there exist polynomial-size RAT proofs of the well-known pigeon hole formulas [HHJW15] while resolution proofs of these formulas are necessarily exponential in size (see Section 3.2)—LQU$^+$-Res cannot simulate QRAT.

On the other hand, QRAT might be able to simulate LQU$^+$-Res, but not with our simulation of the long-distance-resolution calculus, because the simulation cannot convert all LQU$^+$-Res proofs into QRAT proofs. To see this, consider the following QBF $\exists a \forall x \forall y \exists b. (a \lor x \lor b) \land (\bar{a} \lor \bar{x} \lor b) \land (x \lor \bar{b}) \land (\bar{x} \lor \bar{y} \lor \bar{b})$ together with the LQU$^+$-Res proof [BWJ14]: $(x \lor \bar{x} \lor b)$, $(\bar{y} \lor \bar{b})$, $(x \lor \bar{x} \lor \bar{y})$, $(x \lor \bar{x})$, (x), $\bot$. The proof can be illustrated as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{a \lor x \lor b \qquad \bar{a} \lor \bar{x} \lor b}{x \lor \bar{x} \lor b}\,\text{(LQ-res)} \qquad \dfrac{x \lor \bar{b} \qquad \bar{x} \lor \bar{y} \lor \bar{b}}{\bar{y} \lor \bar{b}}\,\text{(QU-res)}}{x \lor \bar{x} \lor \bar{y}}\,\text{(Q-res)}}{x \lor \bar{x}}\,(\forall\text{-red})}{x}\,(\forall\text{-red})}{\bot}\,(\forall\text{-red})$$

In our simulation, we first replace the literal x in $(a \lor x \lor b)$ by x' before resolving the resulting clause $(a \lor x' \lor b)$ with $(\bar{a} \lor \bar{x} \lor b)$. The replacement of x by x' also leads to the addition of $\bar{x}'$ to $(\bar{x} \lor \bar{y} \lor \bar{b})$. If we now perform the universal resolution step of $(x \lor \bar{b})$ with $(\bar{x} \lor \bar{x}' \lor \bar{y} \lor \bar{b})$, then we obtain the following partial proof:

$$\underline{\dfrac{a \lor x' \lor b \qquad \bar{a} \lor \bar{x} \lor b}{x' \lor \bar{x} \lor b}\,\text{(Q-res)} \qquad \dfrac{x \lor \bar{b} \qquad \bar{x} \lor \bar{x}' \lor \bar{y} \lor \bar{b}}{\bar{x}' \lor \bar{y} \lor \bar{b}}\,\text{(QU-res)}}$$

The Q-res step upon b is now impossible because x' is in $(x' \lor \bar{x} \lor b)$ and $\bar{x}'$ is in $(\bar{x}' \lor \bar{y} \lor \bar{b})$. We also cannot eliminate x' from $(x' \lor \bar{x} \lor b)$ via blocked-literal elimination: This would require us to add a new literal x'' to $(x' \lor \bar{x} \lor b)$ and to add $\bar{x}''$ to $(\bar{x}' \lor \bar{y} \lor \bar{b})$ leading to the new pair $x'', \bar{x}''$ of complementary literals.

Our key result, Lemma 56, does not hold anymore when allowing resolution over universal literals. Lemma 56 guarantees that whenever a new literal $\bar{x}'$ is in a proof clause C'_i of the modified long-distance-resolution proof, then $\bar{x}$ was contained in the corresponding

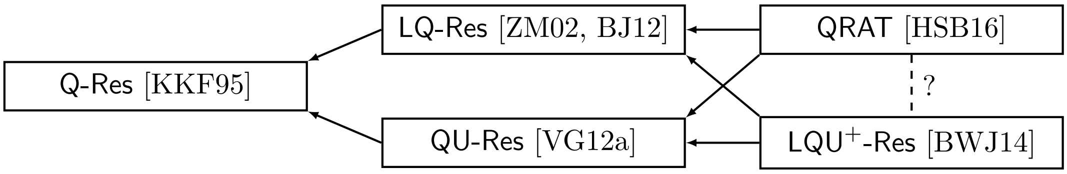


Figure 6.2: Relationship of QRAT with popular resolution-based proof systems. A directed edge from a proof system A to a proof system B indicates that A is strictly stronger than B.

clause C_i in the original proof. The above example shows that resolution over universal literals destroys this property: Although $\bar{x}'$ is contained in the clause $(\bar{x}' \vee \bar{y} \vee \bar{b})$, the literal x is not contained in the corresponding clause $(y \vee \bar{y} \vee b)$ of the original proof because we resolved it away.

CHAPTER 7

Conclusion and Future Work

7.1 Conclusion

We presented several techniques that improve automated-reasoning engines by modifying the syntactic structure of logical formulas.

In the first part of this thesis, we showed that there exist redundancy properties for propositional logic that are more general than blocked clauses while still being local, meaning that they can be decided by considering only the resolution neighborhood of a clause. This locality aspect is part of the reason why blocked clauses have been so successful in the past; it is particularly appealing when dealing with formulas in which the resolution neighborhoods of clauses are small, even if the formulas themselves are big.

By introducing a semantic notion of blocking, we provided the most general local redundancy property. With the aim of bringing this semantic blocking notion closer towards practical SAT solving, we introduced the syntax-based notions of set-blocking and super-blocking. We showed that set-blocked clauses correspond to conditional autarkies, and that super-blocked clauses coincide with semantically blocked clauses.

We then dropped the restriction of locality to obtain even stronger redundancy properties, in particular *propagation-redundant clauses* and restrictions thereof. We introduced these new redundancy properties by first presenting a characterization of clause redundancy that is based on an implication relation between a formula and itself under different partial assignments. Replacing the implication relation in this characterization by efficiently decidable notions of implication, we then obtained various polynomially-checkable redundancy criteria.

We showed that our new redundancy characterization and the corresponding redundancy properties are closely related to other concepts from the literature such as autarkies, variable instantiation, and safe assignments, which we can now capture in a uniform manner. One of our redundancy properties yields a proof system, called DLPR, that coincides with DRAT, which is the de facto standard in SAT solving. Other redundancy properties yield proof systems (SPR and PR as well as their deletion variants DSPR and DPR) that are exponentially stronger than resolution, even if they are not allowed to introduce new variables. We demonstrated this by constructing short proofs without new variables for the well-known pigeon hole formulas. An empirical evaluation shows that our proofs are much smaller than existing clausal proofs and that their correctness can be checked much faster.

To bring these proof systems closer to automated reasoning, we invented a SAT solving paradigm called satisfaction-driven clause learning (SDCL). SDCL generalizes the popular conflict-driven clause learning paradigm by pruning the search space more aggressively. It performs this pruning by learning propagation-redundant clauses, which means that it can produce proofs in the PR proof system. Experiments show that the SDCL solver SaDiCaL can efficiently prove the unsatisfiability of the Tseitin formulas, pigeon hole formulas, and mutilated chessboard problems. Because of theoretical restrictions caused by the weakness of the resolution proof system, CDCL solvers require exponential time to solve these formulas. We therefore believe that SDCL—when combined with sophisticated heuristics and encodings—is a promising SAT-solving paradigm for formulas that are too hard for ordinary CDCL solvers.

In a subsequent part of the thesis, we lifted several popular redundancy properties from propositional logic to first-order logic. To do so, we introduced the principle of implication modulo resolution and its equality variant, the principle of implication modulo *flat* resolution. This allowed us to prove the correctness of the lifted redundancy properties in a uniform way. We also showed how implication modulo flat resolution yields a short soundness proof for the existing technique of predicate elimination [KK16] and we analyzed confluence properties of clause-elimination techniques based on the new redundancy properties. To illustrate the usefulness of these techniques, we implemented one of them—blocked-clause elimination. In an empirical evaluation, we showed that blocked-clause elimination is beneficial for modern provers in many cases, especially when dealing with satisfiable input formulas. Blocked-clause elimination has therefore become a part of the theorem prover Vampire, which is arguably the most efficient theorem prover for first-order logic.

Finally, we used syntactic modification techniques to show that the QRAT proof system polynomially simulates long-distance resolution. In our simulation, we used only a subset of the QRAT rules: Q-resolution, universal reduction, blocked-literal addition, and blocked-literal elimination. Based on our simulation, we implemented a tool that transforms long-distance-resolution proofs into QRAT proofs. The tool allows to merge a QRAT derivation produced by a QBF-preprocessor with a long-distance-resolution proof produced by a search-based solver. The correctness of the resulting QRAT proof can then be checked with a proof checker such as `QRAT-trim` [HSB16]. We evaluated the tool on long-distance-resolution proofs of the Kleine Büning formulas and manually constructed QRAT proofs of these formulas that are smaller than their long-distance counterparts.

7.2 Future Work

Regarding redundancy properties, we plan to lift the notions of set-blocked clauses and propagation-redundant clauses to QBF. As the elimination of redundant clauses has been shown to improve the performance of QBF solvers [BLS11, LBB+15, LE18], we hope that the elimination of set-blocked clauses and propagation-redundant clauses can lead to further performance improvements.

There are still many open questions revolving around our propositional proof systems. In a recent paper, we proved that extended resolution polynomially simulates the DRAT proof system [KRH18]. The combination of this simulation with the simulation that translates DPR proofs into DRAT proofs [HB18] demonstrates that extended resolution polynomially simulates the DPR proof system and therefore also its restricted variants. However, it is an open question how the DPR proof system without new variables relates to other strong proof systems for propositional logic like the polynomial calculus, cutting planes, or even Frege systems. Other open questions are related to the space and width bounds of the smallest DPR proofs, again without new variables. Apart from these theoretical questions, we also want to implement a formally-verified proof checker for DPR proofs.

Although our satisfaction-driven clause learning paradigm can already solve formulas that are too hard for CDCL solvers, it is still outperformed by CDCL solvers on many simpler formulas. This seems to suggest that also in SAT solving, there is no free lunch. However, we believe that the performance of SDCL on simple formulas can be improved by tuning the solver more carefully. For instance, by only learning propagation-redundant clauses when this is really beneficial, or by coming up with a dedicated decision heuristic. To deal with these problems, we are currently investigating an approach based on reinforcement learning.

In the first-order logic part of this thesis, we already hinted at some future work regarding our redundancy properties. Although we have notions of covered clauses and resolution asymmetric tautologies for first-order logic without equality, we still want to provide proper variants for first-order logic with equality. Moreover, the only clause-elimination technique we have implemented so far is blocked-clause elimination. We therefore also want to implement clause-elimination techniques for the other redundancy properties we presented. Implementing a technique for clause *elimination* is arguably easier than implementing clause *addition*. The reason for this is that coming up with useful clauses whose addition boosts prover performance is a non-trivial problem whereas just eliminating existing clauses is straightforward. We therefore also want to spend more efforts on finding beneficial clause-addition techniques in the future.

When it comes to the QRAT proof system, we illustrated that our simulation breaks down if the long-distance-resolution calculus is extended by resolution upon universal literals, as in the calculus LQU^+-Res [BWJ14]. Investigating the exact relationship between LQU^+-Res and QRAT therefore remains open for future work. Another open question is whether or not blocked-literal elimination can be polynomially simulated in LQU^+-Res.

We also don't know if long-distance resolution can be simulated with only Q-resolution, universal reduction, clause deletion, and blocked-literal elimination (but without blocked-literal addition, as in our current simulation). Finally, what is still unclear is how QRAT relates to instantiation-based proof systems and sequent proof systems. Answers to these questions will shed more light on the proof-complexity landscape of QBF.

List of Figures

List of Tables

Index

Bibliography

[ABCH02] Gunnar Andersson, Per Bjesse, Byron Cook, and Ziyad Hanna. A proof engine approach to solving combinational design automation problems. In *Proceedings of the 39th Annual Design Automation Conference (DAC 2002)*, pages 725–730. ACM, 2002.

[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 2010)*. AAAI Press, 2010.

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

[BCJ14] Olaf Beyersdorff, Leroy Chew, and Mikoláš Janota. On unification of QBF resolution-based calculi. In *Proceedings of the 39th International Symposium on Mathematical Foundations of Computer Science (MFCS 2014)*, volume 8635 of *LNCS*, pages 81–93. Springer, 2014.

[BCJ15] Olaf Beyersdorff, Leroy Chew, and Mikolás Janota. Proof complexity of resolution-based QBF calculi. In *Proceedings of the 32nd International Symposium on Theoretical Aspects of Computer Science (STACS 2015)*, volume 30 of *LIPIcs*, pages 76–89. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015.

[BCMS16] Olaf Beyersdorff, Leroy Chew, Meena Mahajan, and Anil Shukla. Are short proofs narrow? QBF resolution is not simple. In *Proceedings of the 33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016)*, volume 47 of *LIPIcs*, pages 15:1–15:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

[BFHB14] Tomáš Balyo, Andreas Fröhlich, Marijn J. H. Heule, and Armin Biere. Everything you always wanted to know about blocked sets (but were afraid to ask). In *Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing (SAT 2014)*, volume 8561 of *LNCS*, pages 317–332. Springer, 2014.

[BG01] Leo Bachmair and Harald Ganzinger. Resolution theorem proving. In *Handbook of Automated Reasoning (in 2 volumes)*, pages 19–99. Elsevier and MIT Press, 2001.

[BHJ17] Tomáš Balyo, Marijn J. H. Heule, and Matti Järvisalo. SAT competition 2016: Recent developments. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI 2017)*. AAAI Press, 2017.

[BJ12] Valeriy Balabanov and Jie-Hong Roland Jiang. Unified QBF certification and its applications. *Formal Methods in System Design*, 41(1):45–65, 2012.

[BLS11] Armin Biere, Florian Lonsing, and Martina Seidl. Blocked clause elimination for QBF. In *Proceedings of the 23rd International Conference on Automated Deduction (CADE 2011)*, volume 6803 of *LNCS*, pages 101–115. Springer, 2011.

[BM08] Marco Benedetti and Hratch Mangassarian. QBF-based formal verification: Experience and perspectives. *Journal on Satisfiability, Boolean Modeling and Computation (JSAT)*, 5(1-4):133–191, 2008.

[BN98] Franz Baader and Tobias Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

[BP16] Olaf Beyersdorff and Ján Pich. Understanding Gentzen and Frege systems for QBF. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2016)*, pages 146–155. ACM, 2016.

[BWJ14] Valeriy Balabanov, Magdalena Widl, and Jie-Hong Roland Jiang. QBF resolution systems and their proof complexities. In *Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing (SAT 2014)*, volume 8561 of *LNCS*, pages 154–169. Springer, 2014.

[CGLR96] James Crawford, Matthew Ginsberg, Eugene Luks, and Amitabha Roy. Symmetry-breaking predicates for search problems. In *Proceedings of the 5th International Conference on Principles of Knowledge Representation and Reasoning (KR 1996)*, pages 148–159. Morgan Kaufmann, 1996.

[Che15] Jingchao Chen. Fast blocked clause decomposition with high quality. *CoRR*, abs/1507.00459, 2015.

[Che16] Hubie Chen. Proof complexity modulo the polynomial hierarchy: Understanding alternation as a source of hardness. In *Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*, volume 55 of *LIPIcs*, pages 94:1–94:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

[Chu36] Alonzo Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58(2):345–363, 1936.

[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. ACM, 1971.

[Coo76] Stephen A. Cook. A short proof of the pigeon hole principle using extended resolution. *SIGACT News*, 8(4):28–32, October 1976.

[CP86] Philip T. Cox and Tomasz Pietrzykowski. Incorporating equality into logic programming via surface deduction. *Annals of Pure and Applied Logic*, 31:177–189, 1986.

[CR79] Stephen A. Cook and Robert A. Reckhow. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44(1):36–50, 1979.

[CS00] Philippe Chatalic and Laurent Simon. Multi-resolution on compressed sets of clauses. In *Proceedings of the 12th IEEE International Conference on Tools with Artificial Intelligence* (*ICTAI 2000*), pages 2–10, 2000.

[DBBD16] Jo Devriendt, Bart Bogaerts, Maurice Bruynooghe, and Marc Denecker. Improved static symmetry breaking for SAT. In *Proceedings of the 19th International Conference on Theory and Applications of Satisfiability Testing* (*SAT 2016*), volume 9710 of *LNCS*, pages 104–122. Springer, 2016.

[DR01] Stefan S. Dantchev and Søren Riis. “Planar” tautologies hard for resolution. In *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science* (*FOCS 2001*), pages 220–229. IEEE Computer Society, 2001.

[EB05] Niklas Eén and Armin Biere. Effective preprocessing in SAT through variable and clause elimination. In *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing* (*SAT 2005*), volume 3569 of *LNCS*, pages 61–75. Springer, 2005.

[Egl16] Uwe Egly. On stronger calculi for QBFs. In *Proceedings of the 19th International Conference on Theory and Applications of Satisfiability Testing* (*SAT 2016*), volume 9710 of *LNCS*, pages 419–434. Springer, 2016.

[ELW13] Uwe Egly, Florian Lonsing, and Magdalena Widl. Long-distance resolution: Proof generation and strategy extraction in search-based QBF solving. In *Proceedings of the 19th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning* (*LPAR-19*), volume 8312 of *LNCS*, pages 291–308. Springer, 2013.

[Fit96] Melvin Fitting. *First-Order Logic and Automated Theorem Proving*. Graduate Texts in Computer Science. Springer, 2nd edition, 1996.

[Göd29] Kurt Gödel. *Über die Vollständigkeit des Logikkalküls*. PhD thesis, University of Vienna, 1929.

[Göd31] Kurt Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte für Mathematik und Physik*, 38(1):173–198, Dec 1931.

[HA28] David Hilbert and Wilhelm Ackermann. *Grundzüge der theoretischen Logik*. Springer, 1928.

[Hak85] Armin Haken. The intractability of resolution. *Theoretical Computer Science*, 39:297–308, 1985.

[HB13] Marijn J. H. Heule and Armin Biere. Blocked clause decomposition. In Ken McMillan, Aart Middeldorp, and Andrei Voronkov, editors, *Proceedings of the 19th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning* (*LPAR-19*), volume 8312 of *LNCS*, pages 423–438. Springer, 2013.

[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 2018*), volume 10806 of *LNCS*, pages 75–92. Springer, 2018.

[Heu18] Marijn J. H. Heule. Schur number five. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence* (*AAAI 2018*). AAAI Press, 2018.

[HHJW15] Marijn J. H. Heule, Warren A. Hunt Jr., and Nathan D. Wetzler. Expressing symmetry breaking in DRAT proofs. In *Proceedings of the 25th International Conference on Automated Deduction* (*CADE 2015*), volume 9195 of *LNCS*, pages 591–606. Springer, 2015.

[HJB10a] Marijn J. H. 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-17*), volume 6397 of *LNCS*, pages 357–371. Springer, 2010.

[HJB10b] Marijn J. H. Heule, Matti Järvisalo, and Armin Biere. Covered clause elimination. In *Short papers for the 17th International Conference on Logic for Programming, Artificial intelligence, and Reasoning* (*LPAR-17-short*), volume 13 of *EPiC Series in Computing*, pages 41–46. EasyChair, 2010.

[HJKW17] Marijn J. H. Heule, Warren A. Hunt Jr., Matt Kaufmann, and Nathan Wetzler. Efficient, verified checking of propositional proofs. In *Proceedings of the 8th International Conference on Interactive Theorem Proving* (*ITP 2017*), volume 10499 of *LNCS*, pages 269–284. Springer, 2017.

[HJL$^+$15] Marijn J. H. Heule, Matti Järvisalo, Florian Lonsing, Martina Seidl, and Armin Biere. Clause elimination for SAT and QSAT. *Journal of Artifical Intelligence Research*, 53:127–168, 2015.

[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 *LNCS*, pages 130–147. Springer, 2017.

[HKB19a] Marijn J. H. Heule, Benjamin Kiesl, and Armin Biere. Encoding redundancy for satisfaction-driven clause learning. In *Proceedings of the 25th Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2019)*, 2019. Accepted for Publication.

[HKB19b] Marijn J. H. Heule, Benjamin Kiesl, and Armin Biere. Strong extension-free proof systems. In *Journal of Automated Reasoning*. Springer, 2019. Accepted for Publication.

[HKKV12] Krystof Hoder, Zurab Khasidashvili, Konstantin Korovin, and Andrei Voronkov. Preprocessing techniques for first-order clausification. In *Proceedings of the 12th Conference on Formal Methods in Computer-Aided Design (FMCAD 2012)*, pages 44–51. IEEE, 2012.

[HKM16] Marijn J. H. Heule, Oliver Kullmann, and Victor W. Marek. Solving and verifying the boolean pythagorean triples problem via cube-and-conquer. In *Proceedings of the 19th International Conference on Theory and Applications of Satisfiability Testing (SAT 2016)*, volume 9710 of *LNCS*, pages 228–245. Springer, 2016.

[HKSB17] Marijn J. H. Heule, Benjamin Kiesl, Martina Seidl, and Armin Biere. PRuning through satisfaction. In *Proceedings of the 13th Haifa Verification Conference (HVC 2017)*, volume 10629 of *LNCS*, pages 179–194. Springer, 2017.

[HS18] Jaakko J. Hintikka and Paul Vincent Spade. History of logic. `https://www.britannica.com/topic/history-of-logic`, 2018. Accessed: 2018-09-28.

[HSB15] Marijn J. H. Heule, Martina Seidl, and Armin Biere. Blocked literals are universal. In *Proceedings of the 7th International NASA Formal Methods Symposium (NFM 2015)*, volume 9058 of *LNCS*, pages 436–442. Springer, 2015.

[HSB16] Marijn J. H. Heule, Martina Seidl, and Armin Biere. Solution validation and extraction for QBF preprocessing. *Journal of Automated Reasoning*, pages 1–29, 2016.

[IMS15] Markus Iser, Norbert Manthey, and Carsten Sinz. Recognition of nested gates in CNF formulas. In *Proceedings of the 18th International Conference on Theory and Applications of Satisfiability Testing (SAT 2015)*, volume 9340 of *LNCS*, pages 255–271. Springer, 2015.

[IP01] Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.

[Jan16] Mikolás Janota. On Q-Resolution and CDCL QBF solving. In *Proceedings of the 19th International Conference on Theory and Applications of Satisfiability Testing* (*SAT 2016*), volume 9710 of *LNCS*, pages 402–418. Springer, 2016.

[JBH10] Matti Järvisalo, Armin Biere, and Marijn J. H. Heule. Blocked clause elimination. In *Proceedings of the 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems* (*TACAS 2010*), volume 6015 of *LNCS*, pages 129–144. Springer, 2010.

[JGMS13] Mikolás Janota, Radu Grigore, and João Marques-Silva. On QBF proofs and preprocessing. In *Proceedings of the 19th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning* (*LPAR-19*), volume 8312 of *LNCS*, pages 473–489. Springer, 2013.

[JHB12] Matti Järvisalo, Marijn J. H. Heule, and Armin Biere. Inprocessing rules. In *Proceedings of the 6th International Joint Conference on Automated Reasoning* (*IJCAR 2012*), volume 7364 of *LNCS*, pages 355–370. Springer, 2012.

[JKMSC16] Mikolás Janota, William Klieber, João Marques-Silva, and Edmund M. Clarke. Solving QBF with counterexample guided refinement. *Artificial Intelligence*, 234:1–25, 2016.

[KB09] Hans Kleine Büning and Uwe Bubeck. Theory of quantified boolean formulas. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 735–760. IOS Press, 2009.

[KHS17] Benjamin Kiesl, Marijn J. H. Heule, and Martina Seidl. A little blocked literal goes a long way. In *Proceedings of the 20th International Conference on Theory and Applications of Satisfiability Testing* (*SAT 2017*), volume 10491 of *LNCS*, pages 281–297. Springer, 2017.

[KK09] Hans Kleine Büning and Oliver Kullmann. Minimal unsatisfiability and autarkies. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 339–401. IOS Press, 2009.

[KK16] Zurab Khasidashvili and Konstantin Korovin. Predicate elimination for preprocessing in first-order theorem proving. In *Proceedings of the 19th International Conference on Theory and Applications of Satisfiability Testing* (*SAT 2016*), volume 9710 of *LNCS*, pages 361–372. Springer, 2016.

[KKF95] Hans Kleine Büning, Marek Karpinski, and Andreas Flögel. Resolution for quantified boolean formulas. *Information and Computation*, 117(1):12–18, 1995.

[KL15] Boris Konev and Alexei Lisitsa. Computer-aided proof of Erdős discrepancy properties. *Artificial Intelligence*, 224(C):103–118, July 2015.

[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 2018*), volume 10900 of *LNCS*, pages 516–531. Springer, 2018.

[KS17] Benjamin Kiesl and Martin Suda. A unifying principle for clause elimination in first-order logic. In *Proceedings of the 26th International Conference on Automated Deduction* (*CADE-26*), volume 10395 of *LNCS*, pages 274–290. Springer, 2017.

[KSS+17] Benjamin Kiesl, Martin Suda, Martina Seidl, Hans Tompits, and Armin Biere. Blocked clauses in first-order logic. In *Proceedings of the 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning* (*LPAR-21*), volume 46 of *EPiC Series in Computing*, pages 31–48. EasyChair, 2017.

[KSTB16] Benjamin Kiesl, Martina Seidl, Hans Tompits, and Armin Biere. Super-blocked clauses. In *Proceedings of the 8th International Joint Conference on Automated Reasoning* (*IJCAR 2016*), volume 9706 of *LNCS*, pages 45–61. Springer, 2016.

[KSTB17] Benjamin Kiesl, Martina Seidl, Hans Tompits, and Armin Biere. Blockedness in propositional logic: Are you satisfied with your neighborhood? In *Proceedings of the 26th International Joint Conference on Artificial Intelligence* (*IJCAI 2017*), pages 4884–4888. ijcai.org, 2017.

[KSTB18] Benjamin Kiesl, Martina Seidl, Hans Tompits, and Armin Biere. Local redundancy in SAT: generalizations of blocked clauses. *Logical Methods in Computer Science*, 14(4), 2018.

[Kul99] Oliver Kullmann. On a generalization of extended resolution. *Discrete Applied Mathematics*, 96-97:149–176, 1999.

[Kul09] Oliver Kullmann. Fundaments of branching heuristics. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 205–244. IOS Press, 2009.

[KV13] Laura Kovács and Andrei Voronkov. First-order theorem proving and Vampire. In *Proceedings of the 25th International Conference on Computer*

Aided Verification, (CAV 2013), volume 8044 of *LNCS*, pages 1–35. Springer, 2013.

[LBB+15] Florian Lonsing, Fahiem Bacchus, Armin Biere, Uwe Egly, and Martina Seidl. Enhancing search-based QBF solving by dynamic blocked clause elimination. In *Proceedings of the 20th International Conference on Logic for Programming, Artificial Intelligence (LPAR-20)*, volume 9450 of *LNCS*, pages 418–433. Springer, 2015.

[LE17] Florian Lonsing and Uwe Egly. DepQBF 6.0: A search-based QBF solver beyond traditional QCDCL. *CoRR*, abs/1702.08256, 2017.

[LE18] Florian Lonsing and Uwe Egly. QRAT+: generalizing QRAT by a more powerful QBF redundancy property. In *Proceedings of the 9th International Joint Conference on Automated Reasoning (IJCAR 2018)*, volume 10900 of *LNCS*, pages 161–177. Springer, 2018.

[Lei97] Alexander Leitsch. *The Resolution Calculus.* Texts in Theoretical Computer Science. Springer, 1997.

[McC64] John McCarthy. A tough nut for proof procedures. *Memo 16, Stanford Artificial Intelligence Project*, July 1964.

[MHB13] Norbert Manthey, Marijn J. H. Heule, and Armin Biere. Automated reencoding of boolean formulas. In *Proceedings of the 8th International Haifa Verification Conference (HVC 2012)*, volume 7857 of *LNCS*. Springer, 2013.

[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 2001)*, pages 530–535. ACM, 2001.

[MPW13] Norbert Manthey, Tobias Philipp, and Christoph Wernhard. Soundness of inprocessing in clause sharing SAT solvers. In *Proceedings of the 16th International Conference on Theory and Applications of Satisfiability Testing (SAT 2013)*, volume 7962 of *LNCS*, pages 22–39. Springer, 2013.

[MS72] A. R. Meyer and L. J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *Proceedings of the 13th Annual Symposium on Switching and Automata Theory (SWAT 1972)*, pages 125–129. IEEE Computer Society, 1972.

[MS85] B. Monien and E. Speckenmeyer. Solving satisfiability in less than 2^n steps. *Discrete Applied Mathematics*, 10(3):287–295, 1985.

[MSLM09] João Marques-Silva, Ines Lynce, and Sharad Malik. Conflict-driven clause learning SAT solvers. In Armin Biere, Marijn Heule, Hans van Maaren, and

Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 131–153. IOS Press, 2009.

[MSS99] João Marques-Silva and Karem A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999.

[Nor15] Jakob Nordström. On the interplay between proof complexity and SAT solving. *SIGLOG News*, 2(3):19–44, 2015.

[NR18] Alexander Nadel and Vadim Ryvchin. Chronological backtracking. In *Proceedings of the 21st International Conference on Theory and Applications of Satisfiability Testing* (*SAT 2018*), volume 10929 of *LNCS*, pages 111–121. Springer, 2018.

[Pel86] Francis Jeffry Pelletier. Seventy-five problems for testing automatic theorem provers. *Journal of Automated Reasoning*, 2(2):191–216, 1986.

[RSV14] Giles Reger, Martin Suda, and Andrei Voronkov. The challenges of evaluating a new feature in Vampire. In *Proceedings of the 1st and 2nd Vampire Workshops* (*Vampire@VSL 2014, Vampire@CADE 2015*), volume 38 of *EPiC Series in Computing*, pages 70–74. EasyChair, 2014.

[Sho78] Robert E. Shostak. An algorithm for reasoning about equality. *Communications of the ACM*, 21(7):583–585, 1978.

[SS14] Friedrich Slivovsky and Stefan Szeider. Variable dependencies and Q-resolution. In *Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing* (*SAT 2014*), volume 8561 of *LNCS*, pages 269–284. Springer, 2014.

[SST14] Aaron Stump, Geoff Sutcliffe, and Cesare Tinelli. StarExec: A cross-community infrastructure for logic solving. In *Proceedings of the 7th International Joint Conference on Automated Reasoning* (*IJCAR 2014*), volume 8562 of *LNCS*, pages 367–373. Springer, 2014.

[SU16] Geoff Sutcliffe and Josef Urban. The CADE-25 Automated Theorem Proving System Competition - CASC-25. *AI Communications*, 29(3):423–433, 2016.

[Sut09] Geoff Sutcliffe. The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0. *Journal of Automated Reasoning*, 43(4):337–362, 2009.

[Tse68] Grigorii Samuilovich Tseitin. On the complexity of derivation in propositional calculus. *Studies in Mathematics and Mathematical Logic*, 2:115–125, 1968.

[Tur37] A. M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265, 1937.

[Urq87] Alasdair Urquhart. Hard examples for resolution. *Journal of the ACM*, 34(1):209–219, 1987.

[VG12a] Allen Van Gelder. Contributions to the theory of practical quantified boolean formula solving. In *Proceedings of the 18th International Conference on Principles and Practice of Constraint Programming* (*CP 2012*), volume 7514 of *LNCS*, pages 647–663. Springer, 2012.

[VG12b] Allen Van Gelder. Producing and verifying extremely large propositional refutations. *Annals of Mathematics and Artificial Intelligence*, 65(4):329–372, 2012.

[WFS06] Sean Weaver, John V. Franco, and John S. Schlipf. Extending existential quantification in conjunctions of BDDs. *Journal on Satisfiability, Boolean Modeling and Computation* (*JSAT*), 1(2):89–110, 2006.

[WHHJ14] Nathan D. Wetzler, Marijn J. H. Heule, and Warren A. Hunt Jr. DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In *Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing* (*SAT 2014*), volume 8561 of *LNCS*, pages 422–429. Springer, 2014.

[ZM02] Lintao Zhang and Sharad Malik. Conflict driven learning in a quantified boolean satisfiability solver. In *Proceedings of the 2002 IEEE/ACM International Conference on Computer-aided Design* (*ICCAD 2002*), pages 442–449. ACM/IEEE Computer Society, 2002.