JOHANNES KEPLER
UNIVERSITY LINZ

Submitted by
Dipl.-Ing.
Katalin Fazekas

Submitted at
Institute for Formal Models and Verification

First Advisor
Univ.-Prof. Dr.
Armin Biere

Second Advisor
Univ.-Prof.
Roderick Bloem, Ph.D.

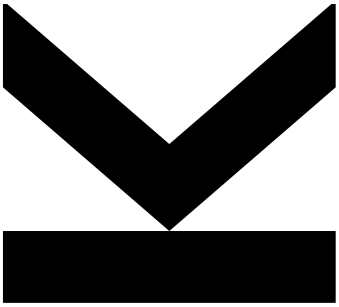
Co-Advisor
Assoc.-Univ. Prof. Dr.
Martina Seidl

Thesis Reviewer
Prof. Laurent Simon

Thesis Reviewer
Prof. Ofer Strichman

June 2020

On SAT-based Solution Methods for Computational Problems



Doctoral Thesis
to obtain the academic degree of
Doktorin der technischen Wissenschaften
in the Doctoral Program
Technische Wissenschaften

JOHANNES KEPLER
UNIVERSITY LINZ
Altenbergerstraße 69
4040 Linz, Österreich
www.jku.at
DVR 0093696

Abstract

Many decision problems, search problems and optimization problems raise questions that can be answered automatically by computers. These computational problems are of great interest in computer science. Deciding the satisfiability of a propositional formula (in short the SAT problem) is a classical NP-complete computational problem that has been studied thoroughly in the last six decades. As a result of this vast research, current SAT solvers are efficiently used in a wide variety of application domains.

The focus of this thesis is on how to exploit these efficient SAT solvers and solving techniques in solution methods for computational problems that are beyond SAT. We model most of the considered methods as conditional transition systems over abstract states. This formalization abstracts away the complex implementation details while it allows to formally reason about invariants, soundness, completeness and termination of the procedures.

First we provide a formal framework to describe and capture how conflict-driven clause learning, the solution technique employed by current SAT solvers, can be applied and extended to address the PSPACE-complete decision problem of quantified Boolean formulas.

Then we focus on the computational task where the SAT problem is extended with some first-order theories (e.g. functions, arrays). Our aim is to find a satisfying solution that optimizes a pseudo-Boolean objective function. We propose a procedure where optimization, propositional reasoning, and theory reasoning are clearly separated. This allows the exploitation of efficient specialized solvers for each of these three components.

Further, we address incremental SAT solving and continuous formula simplifications: two crucial components for efficient computation in several applications. We propose a sound solution method that combines these powerful techniques with less effort and more benefits than before.

Finally, we take a look at the antibandwidth problem, a graph labeling task with an objective to maximize the smallest difference between labels of neighbouring nodes. We introduce a very compact representation of that problem based on binary decision diagrams and then show how to use SAT solvers as efficient black-boxes in an iterative solution method.

Zusammenfassung

Viele Entscheidungsprobleme, Suchprobleme und Optimierungsprobleme werfen Fragen auf, die von Computern automatisch beantwortet werden können. Diese Rechenprobleme sind von großem Interesse in der Informatik. Entscheidung über die Erfüllbarkeit einer aussagenlogischen Formel (d. H. das SAT Problem) ist ein klassisches NP-vollständiges Rechenproblem. In den letzten sechs Jahrzehnten wurde dieses Problem gründlich erforscht. Als Ergebnis dieser umfangreichen Forschung werden aktuelle SAT Löser in einer Vielzahl von Anwendungsbereichen effizient eingesetzt.

Der Schwerpunkt dieser Arbeit liegt auf Lösungsmethoden, die auf diesen effizienten Lösern und Techniken basieren, um bestimmte Rechenprobleme jenseits von SAT zu lösen. Wir modellieren die meisten der betrachteten Methoden als bedingte Übergangssysteme über abstrakte Zustände. Diese Formalisierung lässt die komplexen Implementierungsdetails aus, ermöglicht es jedoch formal über Invarianten, Korrektheit, Vollständigkeit und Beendigung der Verfahren zu argumentieren.

Zunächst presentieren wir einen formalen Rahmen zur Beschreibung wie konfliktgetriebenes Klauselllernen, eine von aktuellen SAT Lösern angewandte Lösungstechnik, angewandt und erweitert werden kann, um das PSPACE-vollständige Erfüllbarkeitsproblem für quantifizierte boolesche Formeln zu adressieren.

Dann konzentrieren wir uns auf das Rechenproblem bei dem das SAT Problem mit erstrangigen Theorien (z. B. Funktionen, Arrays) erweitert wird. Unser Ziel ist es, eine Lösung zu finden, die eine pseudo-Boolesche Zielfunktion optimiert. Wir schlagen ein Verfahren vor, bei dem Optimierung, logisches Schließen und theoretisches Schließen klar getrennt sind. Dies ermöglicht die Nutzung effizienter spezialisierter Löser für jede dieser drei Komponenten.

Darüber hinaus befassen wir uns mit inkrementeller SAT-Lösung und kontinuierlicher Formel Vereinfachungen: zwei wesentliche Komponenten für eine effiziente Berechnung in mehreren Anwendungen. Wir schlagen eine Lösungsmethode vor, die diese leistungsstarken Techniken mit weniger Aufwand und mehr Vorteilen kombiniert als bisher.

Schließlich werfen wir einen Blick auf das Antibandwidth-Problem, ein Graphenmarkierungsproblem mit dem Ziel, die kleinste Differenz zwischen Markierungen von benachbarten Knoten zu maximieren. Wir führen eine sehr kompakte Kodierung dieses Problems, basierend auf binären Entscheidungsdiagrammen, ein und zeigen, wie generische SAT Löser in einer iterativen Lösungsmethode eingesetzt werden können.

Statutory Declaration

I hereby declare that the thesis submitted is my own unaided work, that I have not used other than the sources indicated, and that all direct and indirect sources are acknowledged as references.

This printed thesis is identical with the electronic version submitted.

Parts of this thesis have been published as international conference articles (see [93], [94], [96], and [97] for further details).

Place, date	Signature

Acknowledgements

Attending lectures held by Armin Biere during my exchange semester in Austria has changed the course of my life on many different levels. I will always be grateful for these changes. I am thankful not just for the teaching, the chance, the freedom and the guidance, but also for the great example that Armin sets. And of course, for providing access to the amazing cluster. I want to convey my gratitude also to my co-advisor, Martina Seidl, whom I could always ask and from whom I could learn from the very beginning.

I wish to thank my second advisor Roderick Bloem for his support. Moreover, I thank Ofer Strichman and Laurent Simon for taking the time to review this thesis. I am grateful for the LogiCS doctoral program for providing a frame for research among universities of Austria. I am very thankful for the FORSYTE group at TU Wien for including me as a long-term visitor during my studies. I am grateful for the collaboration with all my co-authors, it was a pleasure and I could learn a lot during it. Further, I am thankful for Florian Lonsing for the discussions and his technical support regarding depQBF.

I would like to thank Fahiem Bacchus for the chance to see the cooler side of the planet by hosting me in Toronto and for his inspiring opinion regarding academic life. I also thank Sheila McIlraith and her group for taking me in and for providing a piece of home in the form of espresso coffee there.

I had the great opportunity to visit Christoph Scholl during my PhD. I am very grateful for the endless kindness of him and his group that made my stay in Freiburg such a nice memory.

I also would like to thank all my past and present colleagues and officemates in Linz and Vienna. I am grateful for all the help and advice, for the friendly environment and for the interesting discussions.

My gratitude goes to my mother, who made my education possible in the first place and who always encouraged me to see the world. I am grateful for the continuous support and home provided by my family. I also want to thank my friends for being my company along the way.

Finally, I would like to thank my partner in crime for being there and for being simply the best.

Contents

Chapter 1

Introduction

Many interesting real-world problems can be decomposed into smaller sub-tasks that are automatically solvable by computers. These smaller tasks are called computational problems. Due to their practical relevance they are of great interest in computer science. Deciding the satisfiability of a propositional formula (in short the SAT problem) is a classical NP-complete computational problem that has been studied thoroughly in the last six decades. As a result of this vast research, current SAT solvers are efficiently used in a wide variety of application domains, ranging over many different fields.

The focus of this thesis is on how to exploit these efficient SAT solvers and solving techniques in solution methods for computational problems that are beyond SAT. For example, solving optimization problems with pseudo-Boolean linear objective functions can employ SAT solvers as black-box NP-oracles. Alternatively, one can adapt the internal search of a SAT solver to address optimization problems over bit-vectors. Searching for a solution or a refutation of a quantified Boolean formula can follow similar approaches as state-of-the-art SAT solvers. Deciding the satisfiability of first-order formulas with respect to certain background theories can efficiently combine SAT solving with specialized theory reasoning. These are just some of many examples where the central role and great potential of SAT solvers in combined procedures is already recognized. In general, understanding the challenges of computational problems that are beyond, but related to, SAT enlarges the range of possibilities that can be exploited by novel solution methods. Thus, our goal is to improve this understanding and thereby support the development of enhanced solution methods.

Our methodology is mostly based on modelling the solution methods as abstract solvers. This is a commonly used technique in order to analyse, compare or reason about different formal methods without defining them with exact algorithms or pseudo-code. That approach considers an abstraction of every possible state of the computation and introduces conditional transition rules to abstract the possible operations to manipulate these states. Different strategies to apply these rules may lead to completely different solution methods. The resulting formal frameworks can be seen as reasoning calculi where every possible derivation captures a possible execution of an implementation on an abstract level. Thus, this formalization abstracts away the complex implementation details while it allows to formally reason about invariants, soundness, completeness and termi-

nation of the procedures. Beyond reasoning formally about methods, we also implement several practical tools and evaluate their performance compared to alternative approaches.

1.1 Background

The following chapters are all self-contained in the sense that they formally define and succinctly introduce the necessary concepts and notations. As an extension to it, the goal here is to give a rather high level, informal description of the underlying concepts and to reveal some similarities and differences among the computational problems that are addressed in this thesis. For a formal and exhaustive introduction of the discussed problems and solution approaches, see for example [151] and [43].

1.1.1 Propositional Satisfiability

The *Boolean satisfiability problem*, also called the SAT problem, is a central concept in this thesis. This computational problem is formulated in propositional logic, a simple fragment of logic that focuses on the logical connections between propositions. A *proposition* in that context means a declarative sentence like "Today is Monday." or "$42 < 3$", that is either true or false, but never both or none. Considering these sentences (also called *atoms*) as building blocks and representing them with *Boolean variables*, we can combine them with *logical connectives* to construct formulas. Common logical connectives are represented here by the symbols $\neg, \Rightarrow, \Leftrightarrow, \wedge, \vee$, expressing their standard meaning, i.e. negation, implication, equivalence, conjunction and disjunction, respectively.

There are some specific forms of formulas that are more interesting for us than others. We say that a propositional formula is in *conjunctive normal form* (CNF) if it is built up as a conjunction of clauses. A *clause* is a disjunction of literals, where a *literal* is either a Boolean variable or the negation of it. Note that any propositional formula can be transformed into CNF relatively easily [198, 227]. The dual of CNF is the *disjunctive normal form* (DNF), where the formula is built up as a disjunction of cubes, where a *cube* is a conjunction of literals. Sometimes we consider and represent clauses and cubes not as disjunctions or conjunctions of literals, but rather as sets of literals. A clause (or cube) that has only a single literal is called a *unit clause* (*unit cube* respectively). Further, an *empty clause* is always false and an *empty cube* is always true.

Assigning a single truth value ($\top$ or $\bot$, representing true or false, respectively) for each Boolean variable determines the truth value of the containing formula. A *truth assignment* can be represented in different ways, for example explicitly as an assignment (e.g. $\{x \leftarrow \top, y \leftarrow \bot\}$, where x and y are Boolean variables) or in shorter form as a set of those literals that evaluate to true under it (e.g. $\{x, \neg y\}$). Each literal in that set is called *satisfied*, while literals evaluating to false are called *falsified*. A *clause is satisfied* by an assignment if at least one of the

contained literals is satisfied, while a *cube is satisfied* only if all literals of it evaluate to true. A CNF *formula under a truth assignment* is the formula where every satisfied clause is removed and from the remaining clauses every falsified literal is deleted. A formula in CNF is *satisfiable* if there exists a truth assignment such that it satisfies all clauses.

To illustrate these introduced definitions, consider the CNF formula $\mathcal{F} = (x \vee \neg y) \wedge (\neg x \vee y)$. In total, four possible truth value combinations can be assigned to the two variables of it (x and y). Figure 1.1 depicts each of these truth assignments organized into a tree structure. Each node of the tree is a formula under a certain truth assignment and each edge assigns a truth value for one of the variables. Notice that the variables are assigned in the same order on each path from root to leaf and on each dashed edge a $\bot$ value is assigned. Starting from $\mathcal{F}$ under the empty truth assignment in the root node of the tree,

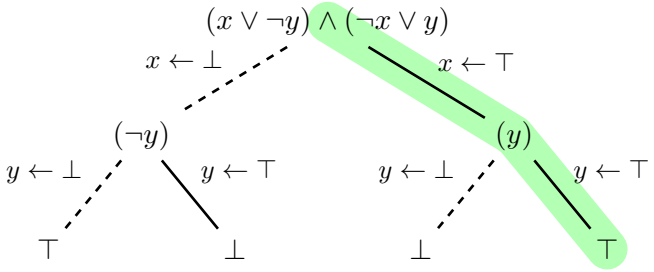


Figure 1.1: Formula $\mathcal{F} = (x \vee \neg y) \wedge (\neg x \vee y)$ under truth assignments to variables x and y. A path of that tree that satisfies $\mathcal{F}$ is highlighted with green.

we first consider the two possible truth values of x. In case it is assigned false (left branch on Figure 1.1), the second clause of $\mathcal{F}$ becomes satisfied, and in the first clause one literal becomes falsified. Thus, $\mathcal{F}$ under the assignment $\{x \leftarrow \bot\}$ is the formula consisting of the unit clause $(\neg y)$, as it is shown on Figure 1.1. Assigning y to true would falsify this clause and thus would make $\mathcal{F}$ false ($\bot$ leaf in Fig. 1.1). Assigning y to false on that branch satisfies the remaining clause, i.e. the assignment $\{x \leftarrow \bot, y \leftarrow \bot\}$ is a solution (also called a *model*) of the formula $\mathcal{F}$. Another model of the formula, where both x and y are assigned true is highlighted with green in Figure 1.1.

Deciding the satisfiability of a CNF formula was shown to be an NP-complete problem already several decades ago [76]. And so in theory it is an intractable problem. Nevertheless, in practice there are many tools that can solve problems over millions of Boolean variables in a few seconds. Thus, several practical problems are solved by SAT solvers in industry. To name just a few of them, SAT solvers are used for hardware and software design and verification [170,201,231], for test pattern generation [156], for planning [68, 144] and for configuration management [132]. The theoretical relevance of that problem is probably shown the best by the observation that the so far longest section of "The Art of Computer Programming", the fundamental computer science book series written by

Donald Knuth, is concerned solely with the Boolean satisfiability problem [147].

1.1.2 Quantification

Allowing to explicitly quantify *existentially* ($\exists$) or *universally* ($\forall$) the variables of a propositional formula leads to the concept of *Quantified Boolean Formulas* (QBF). And with that new concept comes a new decision problem that gives a twist to the question of SAT. In SAT we asked whether there is at least one solution, i.e. an arbitrary satisfying truth assignment to the variables of a given propositional formula. The new task is to find a certain set of truth assignments to the Boolean variables of a propositional formula such that these assignments together fulfil some requirements. More precisely, we look for several models of the formula such that every possible truth-value combination of the universally quantified Boolean variables occurs in the found solutions and every existentially quantified variable can be expressed as a Boolean function over a specific subset of the universally quantified variables. And so here we ask not simply the existence of a single solution, but rather seek a specific combination of several solutions. At the end, this set of solutions can be organized into a tree structure, also called *tree model*, where at each node the truth value of a variable of the formula is decided and the order of these decisions is determined by the quantification of the problem. This tree will be actually a sub-tree of the assignment tree that we introduced for illustration in the previous section in Figure 1.1.

This representation of solutions is easier to understand if we interpret QBFs as a game between two players (see e.g. [146]). There is one player to control the existentially quantified variables and there is another one that is responsible for the universally quantified variables. The order of the players is determined by the order of the quantifiers at the beginning of the formula. The existential player would like to satisfy each clause of the formula, while the universal player always tries to falsify at least one of the clauses. Thus, the previously described tree models are also called the *winning strategy* of the existential player [103], since they show to the existential player how to choose a value for the variables based on the choices of the universal player on each branch. Similarly, for false formulas one can construct a winning strategy for the universal player, showing how to pick a value for the universally quantified variables, based on the current choices of the existential player, such that at least one clause is falsified on each path. This tree is called a *tree refutation* or *counter-model* of the formula.

The following example illustrates a tree model built for a very simple QBF. Consider the quantified Boolean formula $\forall x \exists y.(x \vee \neg y) \wedge (\neg x \vee y)$ and see Figure 1.2 that depicts the solution of it by highlighting those assignments (paths) that belong to it. We already saw in the previous section that the formula $(x \vee \neg y) \wedge (\neg x \vee y)$ has two possible solutions. Now considering the quantification of the variables, we see that these assignments are both needed to construct one solution to the QBF problem. So the solution of the QBF $\forall x \exists y.(x \vee \neg y) \wedge (\neg x \vee y)$ contains two satisfying truth assignments of the formula $(x \vee \neg y) \wedge (\neg x \vee y)$, one

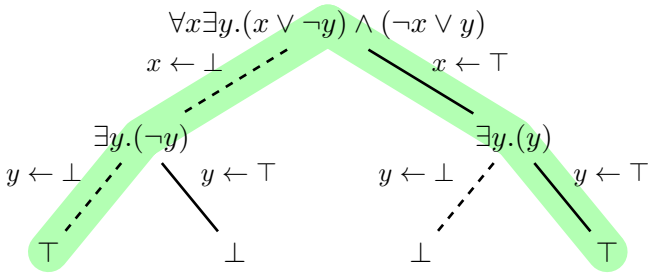


Figure 1.2: Tree model of formula $\forall x \exists y.(x \vee \neg y) \wedge (\neg x \vee y)$ is highlighted with green.

where x is true (right branch) and another where x is false (left branch). Notice that each branch of that model can be written as a cube, which are $\neg x \wedge \neg y$ and $x \wedge y$. Also notice that every possible choice of the universal player belongs to a path from the root to a leaf of the tree. The choice of the existential player in both assignments is to assign the same truth-value to y as x. Thus, the winning strategy of the existential player could be described as a function of variable x; for example as $(y = x \,?\, \top : \bot)$.

One can similarly construct a tree refutation (i.e. a winning strategy for the universal player) in case of unsatisfiable (i.e. false) quantified Boolean formulas. Consider the quantified formula $\exists y \forall x.(x \vee \neg y) \wedge (\neg x \vee y)$, that is very similar to the previous example, but the order of the quantified variables is flipped. In that case, the existential player must make the first decision, and whatever choice it makes, the universal player can always pick a value for x s.t. one of the two clauses is falsified. This is presented on Figure 1.3, where, again, all possible truth assignments of the variables are shown, and the winning strategy of the universal player is highlighted with red. In that example we could summarize this strategy as a Boolean function that takes y as a parameter, and assigns the opposite value of it to x (e.g. as $(x = y \,?\, \bot : \top)$).

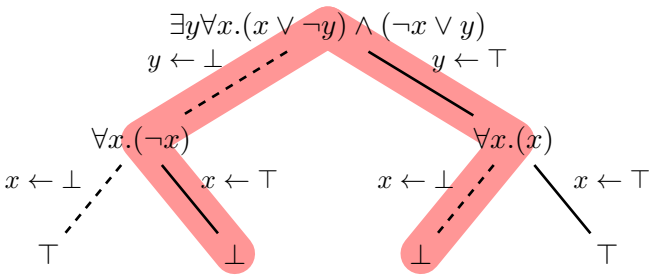


Figure 1.3: Tree refutation of formula $\exists y \forall x.(x \vee \neg y) \wedge (\neg x \vee y)$ is highlighted with red.

Our examples here are of course rather trivial and in practice much larger problems are tackled. Application domains for QBFs are for example planning, synthesis and model checking (see e.g. [30, 219] for surveys of applications).

1.1.3 Background Theories

What happens when the subject of our reasoning is slightly more complicated or more abstract than what we can express with sentences like "Today is Monday."? For example, consider the statements "$s = t$","$t = u$" and "$s \neq u$", where we use (for the sake of this example integer) variables s, t and u. Beyond these *non-Boolean variables*, some *relations* between them are defined with the mathematical symbol "=" and the negation of it. The standard interpretation of the equals sign is well-known, in principle it expresses that two objects are indistinguishable from each other. Mathematically it is a reflexive, symmetric and transitive congruence relation. But these well-known properties remain in the background and they are not written explicitly anywhere in our formulas.

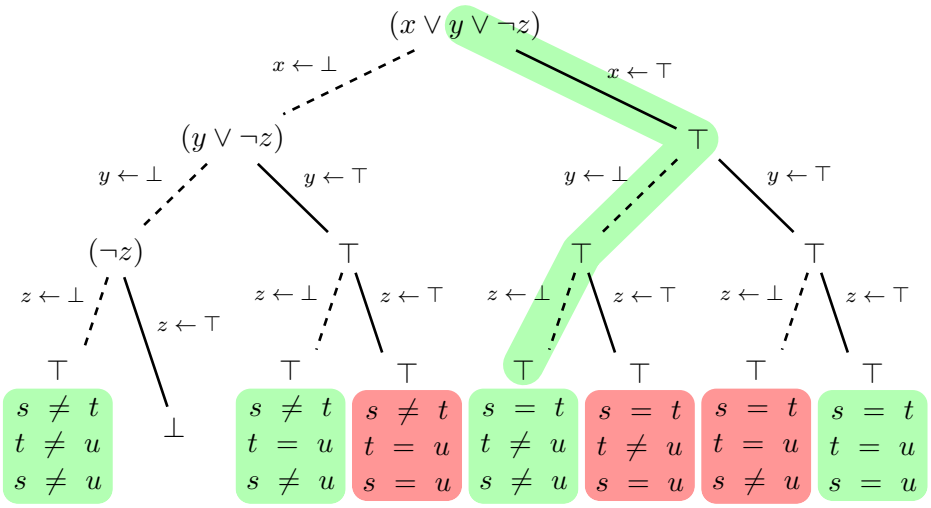


Figure 1.4: An example solution of the SMT formula $(s = t \vee t = u \vee s \neq u)$. The Boolean skeleton of the problem is the propositional formula $(x \vee y \vee \neg z)$, where variable x stands for "$s = t$", y for "$t = u$" and z for "$s = u$".

If we would like to describe the disjunction of these statements in propositional logic, it could be started by the simple formula $x \vee y \vee \neg z$, where the Boolean variable x stands for the truth value of "$s = t$", y for "$t = u$" and z for "$s = u$". This partial representation of the problem, called the *Boolean skeleton*, completely neglects the meaning and the properties of equivalence but grasps the logical connections between the statements of our problem (i.e. of the formula $(s = t \vee t = u \vee s \neq u)$). Moreover, in this abstraction 7 from the 8 possible truth assignments to variables x, y and z satisfy the formula. However, assigning a truth value to one of the Boolean variables implicitly assigns a truth value to the equality which it encodes. And thus every solution of the disjunction $(x \vee y \vee \neg z)$ naturally translates to a 3-long conjunction of equalities and disequalities. Considering our background knowledge, like the domains of the variables s, t and u and the properties of the equality relation (e.g. symmetry and transitivity),

some of these conjoined equalities and disequalities can not be true at the same time. For example, assigning true to the equalities $s = u$ and $t = u$ implies that s and t must be equal (due to the transitivity of equality), and thus in that case the equality $s = t$ is not allowed to be assigned false.

Figure 1.4 depicts all possible truth assignments over the Boolean variables x, y, z. Under each satisfying truth assignment it presents the set of equalities and disequalities (also called *theory literals*) that the assignment translates to. The background color of each of these sets indicates whether the given set of theory literals is consistent with the notion of equality, i.e. whether the properties reflexivity, symmetry and transitivity are maintained. Notice that from the 7 SAT solutions only 4 translated to a consistent set.

A solution for a *satisfiability modulo theories* (SMT) problem can be seen as a solution for the pure SAT abstraction of the problem, such that the assignment translates to a *consistent set of theory literals*. The consistency of a set of theory literals is determined by the *axioms* of the underlying background theory (or set of background theories). In our example this background theory was the simplest, most essential, so called *equalities over uninterpreted functions* (EUF) theory, but in practice there is a wide range of possibilities for background theories and for their combinations. Many solvers support for example the theory of linear (sometimes even non-linear) arithmetic over rational and integer numbers, bit-vectors, arrays, or their combinations. See for example [215] or [27] for a more detailed description of these systems.

What is important to observe here is that SMT problems naturally decompose into two subproblems. First, we need to consider the Boolean abstraction of the formula as a SAT problem. Second, we need to determine efficiently the theory consistency of any set of theory literals built from truth assignments over the variables of this Boolean skeleton. In practice this decomposition is the main organizing principle behind SMT solver implementations. The theory consistency check is the responsibility of the so called *theory solvers* (one dedicated for the conjunctive fragment of each background theory) which are closely collaborating with a SAT solver during SMT solving.

The resulting combined tools are important assets in many practical fields. Formal methods participate in the daily process of many industrial companies in the form of SMT solvers. For example, Amazon uses them to verify properties of access policies [17], Facebook employs them for static analysis of their code base [66] and Microsoft builds on them for efficient debugging [110].

1.1.4 Optimization

In case of SMT solving, one basically has to distinguish "good" SAT solutions from "bad" SAT solutions, depending on whether it translates to a consistent set of theory literals or not. In many computational problems there is in fact a fine-grained range between "good" and "bad" solutions (independently from any background theory) and this quality is usually quantitatively measurable.

Thus, in an *optimization problem* our objective is to find the *"best" solution*, where "best" is identified based on this quantitative measurement.

The purest extension of SAT with optimization can be formulated such that we can compare solutions with each other by simply counting the number of satisfied literals in a predefined set of literals. The more of these literals are satisfied, the "better" the found SAT solution. For example, consider the following formula

$$\mathcal{F}_o = (a_1 \vee \neg x \vee \neg y) \wedge (a_2 \vee x) \wedge (a_3 \vee y)$$

where the number of satisfied literals from the set $\mathcal{A} = \{\neg a_1, \neg a_2, \neg a_3\}$ are the ones that determine the quality of any found SAT solution. Figure 1.5 depicts each possible model (i.e. satisfying truth assignment) of $\mathcal{F}_o$, together with the measurement how "good" each of them is w.r.t. $\mathcal{A}$. While in SMT each SAT solution was mapped to a conjunction of theory literals, in this optimization problem each SAT solution translates to a number. Notice that while the SAT problem $\mathcal{F}_o$ has 14 possible solutions, only three of them have the optimal value of 2 (there is no SAT solution where all three literals of $\mathcal{A}$ are satisfied). One of these *optimal solutions* is highlighted with green in Figure 1.5.

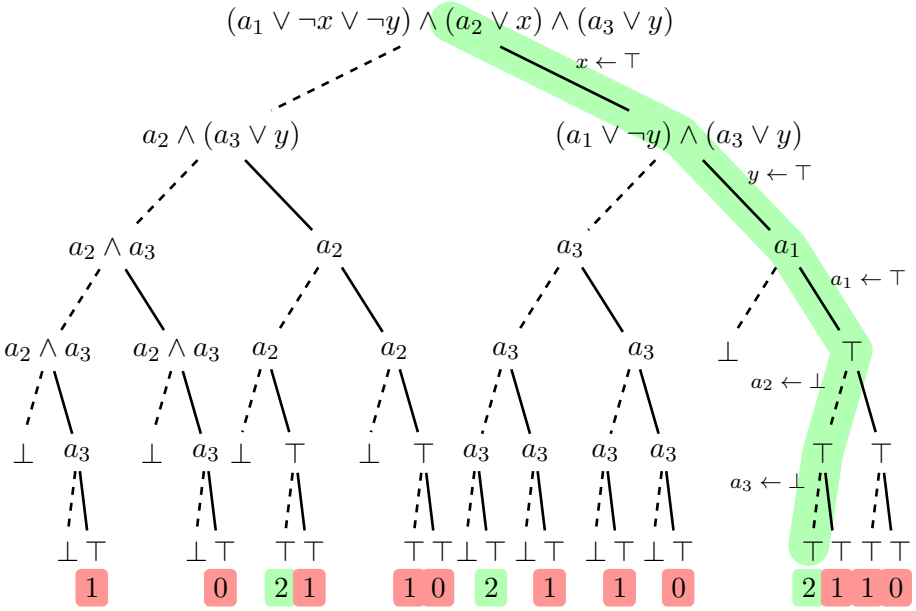


Figure 1.5: An example solution of the optimization problem $(a_1 \vee \neg x \vee \neg y) \wedge (a_2 \vee x) \wedge (a_3 \vee y)$ with the objective to maximize the number of satisfied literals in the set $\{\neg a_1, \neg a_2, \neg a_3\}$. Although the variable assignments are marked explicitly only on the solution path, the order of the variables is the same on every branch.

In that example we measured the quality of solutions (i.e. our *objective value*) based only on the number of the satisfied literals of $\mathcal{A}$ in each SAT solution. In general, the *maximum satisfiability problem* (MaxSAT) maximizes the number of satisfied clauses in a formula among each truth assignment over its variables.

However, one can always construct a set $\mathcal{A}$ by introducing a new literal into each clause and adding it's negation to $\mathcal{A}$. Nevertheless, it is important to see that while MaxSAT usually deals with unsatisfiable formulas, extending it with a counter set $\mathcal{A}$ makes it trivially satisfiable, which allows us to represent the problem as filtering found SAT solutions. This becomes an important feature when we try to combine MaxSAT with other problems (e.g. with reasoning w.r.t. some background theories).

Another important aspect of computational optimization problems is the complexity of the *objective function*, i.e., the subject of our maximization (or minimization) attempt. In our previous example we aimed to maximize the sum of values based on some Boolean variables. This is a simple, linear pseudo-Boolean function. More general optimization problems might consider non-linear objective functions or functions over non-Boolean variables and terms. Our focus in this thesis is mostly on simpler objective functions, but even these problems can occur in several practical domains, such as planning [236], fault localization and debugging [71], package management [8], automated type inference in code analysis [117] or in Bayesian network structure learning [31].

1.1.5 Incremental Problems

Sometimes computational problems do not come in single. For example, in the context of bounded model checking or in planning, instead of having a single SAT problem, a sequence of similar SAT problems need to be solved. During solving a satisfiability problem, a solver learns many details about the formula, like hidden consequences of the clauses or obvious dead-ends of the derivation. A great portion of this learned information continues to hold in a similar formula that is somehow related with this solved problem. In case these related problems are defined explicitly as incremental problems, a solver can exploit and utilise these details to speed up derivations.

An *incremental satisfiability problem* [131] can be seen as a sequence of formulas, such that each formula extends the previous one with a set of constraints. Thus, in each step the problem becomes larger, but a great part of the formula is always such that it was previously already seen and solved by the solver. *Counterexample guided abstraction refinement* (CEGAR, see e.g. [75]) is a common solution technique, used not just in model checking, that heavily relies on these kind of continuous formula extensions.

Another aspect of incrementality that is supported by SAT (and SMT) solvers is the concept of *assumptions* [126, 155]. Assumptions can be seen as temporal unit clauses that are fixing the truth values of some of the variables in the problem. Solving a *SAT problem under assumptions* means that we are looking for a satisfying truth assignment where the assumptions already fixed some values. It might be that a satisfiable SAT problem is unsatisfiable under certain assumptions. Nevertheless, solving the same formula but under different assumptions can reuse many previous efforts.

In practice these two kinds of incremental problems interleave, i.e. formulas are continuously extended with new constraints and they are evaluated multiple times under different sets of assumptions. This combination also allows to deactivate some previously added constraints (by satisfying them with assumptions) and still benefit from incremental solving. Combined procedures that are involving SAT solvers as black-box components usually exploit and build on these incremental possibilities of the solver (see e.g. in model checking [40,52,87,89,154,225], planing [78,109] and in unsatisfiable core enumeration or minimization [178,183]). Thus, support of efficient incremental solving is essential in any state-of-the-art SAT solver.

1.2 Outline

This thesis is split up into three main parts. This chapter constitutes the first part and it is concerned with the conceptual cornerstones and structure of this dissertation. In the previous section we informally introduced the computational problems that are addressed in the upcoming chapters. In this section we shortly describe our motivation behind our work while we overview the remaining chapters of this thesis.

The next part (Chapter 2-5) consists of four peer-reviewed and published papers where the author of this thesis is the main author. The presentation of each publication begins with a short bibliographical description. The style of each paper is modified such that they follow a unified format. These modifications affect not just the representation of the texts (such as fonts, spacing, layout), but the numbering of figures, definitions, theorems, tables and citations.

It is helpful to organize the subjects of the publications and discussions of the following chapters around the computational problems that they attempt to address. Figure 1.6 visualizes it explicitly by connecting the problems and chapters of this thesis. The central concept of the discussed computational problems, as it can be seen in Figure 1.6, is the Boolean satisfiability problem.

In Chapter 2 we consider this problem in combination with Boolean quantification, more precisely we focus on quantified Boolean formulas. In the presented paper [96] we provide an abstract calculus to capture and formally describe duality-aware search-based QBF solvers. The duality-awareness in that context means that these solvers are considering and handling conflict clauses and solution cubes as duals of each other (see e.g. [112,113,210,237]). Although current search-based QBF solvers are not that symmetric and are rather organized around clauses (e.g. the problems are defined only in a clausal form), the presented work provides important theoretical insights. Cubes in the derivations of QBF solvers, especially in the case of true formulas, are essential and the better understanding of their role provides more possibilities to exploit them. In Section 6.1 we describe a line of work that builds on our findings from [96].

In Chapter 3 we present our paper [93] where the Boolean satisfiability prob-

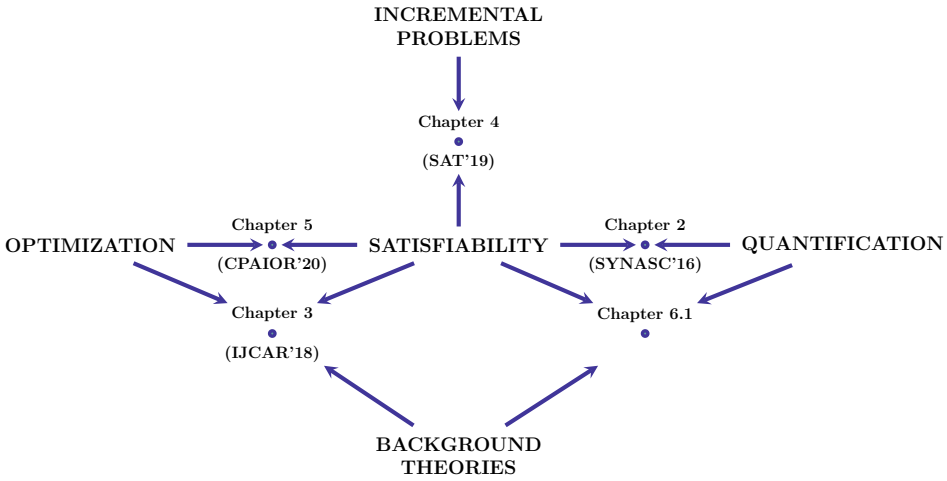


Figure 1.6: Relation between chapters of this thesis and the addressed problems.

lem is extended with theory reasoning and, at the same time, with a pseudo-Boolean objective function to optimize. Though both SMT and MaxSAT are well studied subjects, their combination, where the objective function is defined completely in the Boolean layer, is rarely addressed explicitly. To the best of our knowledge, this paper is the first to define an abstract, assumption-based MaxSAT (and MaxSMT) solver. Extensions of the DPLL(T) framework to describe optimization modulo theories were presented previously (for example in [192]), but not with the focus on the optimization process from the SAT perspective. Further, our described solution instantiates the so-called Implicit Hitting Set (IHS) approach [69, 181, 211], though the presented abstract solver is relatively easy to adapt for other MaxSAT solving approaches. The IHS approach is a general way to address combinatorial optimization problems. Lifting it to the context of MaxSMT allows to completely separate Boolean and theory reasoning from optimization and so has the potential to lead to improved and more robust solvers. An observation of that work is that while most MaxSAT solving methods can be used "as is" to address MaxSMT, an efficient approach must be flexible enough to consider the difference in the relative costs between calling an SMT or a SAT oracle.

Chapter 4 contains our paper [94] where the focus is on improving SAT solving in the context of incremental problems. Incremental SAT solving and continuous formula simplifications (i.e. inprocessing) are two crucial components of efficient tool chains in several applications. In this paper we proposed a sound solution, in the form of a calculus, that allows one to efficiently combine these two powerful techniques with less effort and more benefits than before. As a side-effect, our proposed solution simplifies the way how incremental solvers can be used, by eliminating the necessity to manually identify reoccurring variables in problem encodings. This paper won the best student paper award of the SAT conference in 2019 and significantly shaped the state-of-the-art in incremental SAT solving.

Another optimization problem is addressed in Chapter 5, where our latest paper [97] is presented. In that work we consider the so-called antibandwidth problem, a graph labeling problem with an objective to maximize the smallest difference between numerical labels of neighbouring nodes. We introduced a binary decision diagram (BDD) based SAT encoding with linear size for the at-most-one staircase constraints (called at-most-one sequence constraints in the constraint programming (CP) literature [57, 229]). Thereby we can exploit a SAT solver to efficiently answer the feasibility questions of an integer programming (IP) problem. Sequence constraints are frequently employed in problem representations of CP. Thus, providing a small and efficient alternative encoding for the at-most-one sub-case has a wide application domain. Beyond that, this work illustrated that considering multiple BDDs simultaneously over the same problem but with different variable orderings has great potential to gain compact problem representations.

The last part of this thesis (Chapter 6-7) reflects on, extends and shortly concludes the presented work. In Chapter 6 we extend each of the published work with further details and results and we also succinctly present some unpublished work in progress. In the last chapter (Cahpter 7) we conclude the thesis, summarize our contributions and discuss the current limitations and potential future works based on our findings.

Chapter 2

A Duality-Aware Calculus for Quantified Boolean Formulas

Published

In Proceedings of the 18th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2016), pages 181-186, Timisoara, Romania. See [96].

Authors

Katalin Fazekas, Martina Seidl and Armin Biere.

Abstract

Learning and backjumping are essential features in search-based decision procedures for Quantified Boolean Formulas (QBF). To obtain a better understanding of such procedures, we present a formal framework, which allows to simultaneously reason on prenex conjunctive and disjunctive normal form. It captures both satisfying and falsifying search states in a symmetric way. This symmetry simplifies the framework and offers potential for further variants.

2.1 Introduction

Quantified Boolean Formulas (QBF) extend the language of propositional logic by quantifiers over the propositional variables. In consequence, the decision problem becomes PSPACE-complete making QBF solving interesting for many applications in verification, synthesis, artificial intelligence, etc. (see [30] for a survey). The combination of conflict-driven clause and solution-driven cube learning (QCDCL, see e.g., [108,239]) is the most successful approach for search-based QBF solving [105, 162], lifting conflict-driven clause-learning (CDCL), originating in SAT [176], to QBF. However, if a QBF solver only gets a QBF in prenex conjunctive normal form (PCNF) as input, the search is biased towards conflicts. This asymmetry impedes the whole search process. To overcome this asymmetry, *duality-aware QCDCL solving* considers not only a representation of the input in PCNF but also in prenex disjunctive normal form (PDNF), treating solution and conflict states symmetrically [237]. In [113] it was shown that duality-aware reasoning can easily be added to PCNF-based QCDCL solvers. This paper gives a concise characterization of the behavior of such solvers exploiting the symmetry in the search for conflicts and solutions.

Related Work

The seminal paper of Nieuwenhuis, Oliveras, and Tinelli [194] introduced a rule-based calculus to concisely model the classical CDCL approach for SAT. This work forms the basis of many theoretical investigations on SAT and SMT solving, for instance the recent formalization of CDCL-SAT solving in Isabelle [47]. For QBF, however, we are not aware of any similar rule-based formulation of QCDCL. The abstract QBF solver presented rather informally in [58] lacks non-chronological backtracking (*backjumping*) and learning, i.e., the essential rules of QCDCL. The literature on QCDCL (e.g., [108, 239]) is missing a precise formalization too. In [133] the focus is on the relation of QCDCL with Q-resolution instead of capturing the search precisely.

Outline

We introduce a new calculus which formally captures the behavior of QCDCL solvers. To this end, we first provide generic rules defining a state transition system. Then we introduce a strategy that describes how to apply these rules to get a duality-aware QBF solver which is correct and terminates. Finally, we relate our calculus to standard PCNF QCDCL solvers.

2.2 Preliminaries

A propositional formula over variables $\mathcal{V}$ is in conjunctive normal form (CNF) if it is a conjunction of clauses. A *clause* is a disjunction of literals and a literal

is a variable or its negation. A propositional formula is in disjunctive normal form (DNF) if it is a disjunction of cubes, where a *cube* is a conjunction of literals. If literal ℓ is v or $\neg v$, then $var(\ell) = v$. A *quantified Boolean formula* (QBF) $\mathcal{Q}(\varphi)$ consists of the propositional matrix φ over $\mathcal{V}$ and the quantifier prefix $\mathcal{Q} = Q_1 V_1 \dots Q_n V_n$ where V_i are disjoint sets of variables, $Q_i \in \{\exists, \forall\}$, and $Q_i \neq Q_{i+1}$. In this paper, we assume $\mathcal{V} = \bigcup V_i$, i.e., all variables of the matrix are quantified. Such QBFs are called closed. If φ of QBF $\mathcal{Q}(\varphi)$ is in CNF (DNF), then $\mathcal{Q}(\varphi)$ is in prenex CNF (DNF), denoted PCNF (PDNF). We also write $\ell_\forall$ and $\ell_\exists$ for a universal or existential literal. In that way, the prefix $\mathcal{Q}$ defines a partial ordering relation $<_\mathcal{Q}$ between variables v_i and v_j such that $v_i <_\mathcal{Q} v_j$ if $level(v_i) < level(v_j)$. This ordering is extended to the literals over the variables, that is, $\ell_i <_\mathcal{Q} \ell_j$ if $var(\ell_i) <_\mathcal{Q} var(\ell_j)$. An assignment A is a consistent list of literals which defines a mapping from literals to truth values as follows. If $v \in A$ then v is true under A, if $\neg v \in A$ then v is false under A. An assignment A is called total assignment of $\mathcal{V}$ if every $v \in \mathcal{V}$ is assigned by A, otherwise it is called partial. Occasionally we interpret assignments as cubes. Given an assignment A and a CNF φ with a fixed quantifier prefix $\mathcal{Q}$, we denote by $\varphi[A]$ the CNF under assignment A, where all clauses containing ℓ are removed and all occurrences of $\neg\ell$ are deleted (and $\mathcal{Q}$ is unchanged). For DNF φ, $\varphi[A]$ is defined dually. The negation of a quantifier prefix $\mathcal{Q}$ (denoted with $\neg\mathcal{Q}$) is the simultaneous substitution of $\forall$ quantifiers for $\exists$ quantifiers and vice versa in $\mathcal{Q}$. We define tree models and tree refutations of QBFs as in [103]. We denote an empty clause or cube by $\emptyset$, an empty CNF (DNF) by $\top$ ($\bot$). A QBF $\forall x\mathcal{Q}(\varphi)$ is true iff $\mathcal{Q}(\varphi)[x]$ and $\mathcal{Q}(\varphi)[\neg x]$ are true. A QBF $\exists x\mathcal{Q}(\varphi)$ is true iff $\mathcal{Q}(\varphi)[x]$ or $\mathcal{Q}(\varphi)[\neg x]$ is true. Two QBFs ψ_1 and ψ_2 are equivalent (written as $\psi_1 \equiv \psi_2$) if they have the same truth value.

Definition 2.2.1. *QBFs $\mathcal{Q}(\mathcal{C})$ in PCNF and $\mathcal{Q}(\mathcal{D})$ in PDNF have the* duality property *if $\mathcal{Q}(\mathcal{C}) \equiv \mathcal{Q}(\mathcal{D})$.*

The duality property holds under assignment A, if $\mathcal{Q}(\mathcal{C}[A]) \equiv \mathcal{Q}(\mathcal{D}[A])$. Based on the duality property, dual search-based solvers [113, 237] use both a CNF as well as a DNF of the input QBF (as explained e.g. in [237]) to treat conflicts and solutions symmetrically. Note that in that case the introduced Tseitin variables are existentially quantified in the CNF, and universally quantified in the DNF encoding. Therefore, the joint prefix $\mathcal{Q}$ of CNF and DNF has some variables that occur only in the CNF matrix (the existential Tseitin variables) and some that occur only in the DNF matrix (the universal Tseitin variables).

Definition 2.2.2. *Given QBF $\mathcal{Q}(\varphi)$ in PCNF and a clause C, we define $\varphi \models_\mathcal{Q} C$ to hold if $\mathcal{Q}(\varphi \wedge C) \equiv \mathcal{Q}(\varphi)$.*

Definition 2.2.3. *Given QBF $\mathcal{Q}(\varphi)$ in PDNF and a cube C, we define $\varphi \models^\mathcal{Q} C$ to hold if $\mathcal{Q}(\varphi \vee C) \equiv \mathcal{Q}(\varphi)$.*

Note that $\models_{\mathcal{Q}}$ and $\models^{\mathcal{Q}}$ define the entailment relation w.r.t. prefix $\mathcal{Q}$. Standard propositional entailment is denoted by $\models$. Immediately from the definitions we obtain:

Lemma 2.2.1. *Let $\mathcal{Q}(\varphi)$ be a closed QBF in PCNF and C a clause. Then $\varphi \models_{\mathcal{Q}} C$ iff $\neg\varphi \models^{\neg\mathcal{Q}} \neg C$.*

In the following we fix a quantifier prefix $\mathcal{Q}$ together with the ordering $<_{\mathcal{Q}}$ (on all variables and literals).

Definition 2.2.4. *The* universally tailing literals $\mathcal{T}_\forall^{\mathcal{Q}}(C)$ *of clause C are* $\{\ell_\forall \in C \mid \ell_\exists <_{\mathcal{Q}} \ell_\forall$ *for all* $\ell_\exists$ *in* $C\}$.

Definition 2.2.5. *The* universal reduction *of a clause C is defined as* $\mathcal{R}_\forall^{\mathcal{Q}}(C) = C \setminus \mathcal{T}_\forall^{\mathcal{Q}}(C)$.

Analogously, existential reduction $\mathcal{R}_\exists^{\mathcal{Q}}(C)$ removes the tailing existential literals $\mathcal{T}_\exists^{\mathcal{Q}}(C)$ from cube C. Universal (existential) reduction can be extended to a set of clauses (cubes) $\mathcal{C}$ as $\mathcal{R}_\forall^{\mathcal{Q}}(\mathcal{C}) = \{\mathcal{R}_\forall^{\mathcal{Q}}(C) \mid C \in \mathcal{C}\}$ ($\mathcal{R}_\exists^{\mathcal{Q}}(\mathcal{C}) = \{\mathcal{R}_\exists^{\mathcal{Q}}(C) \mid C \in \mathcal{C}\}$). W.l.o.g. we assume clauses (cubes) of the input QBFs to be non-tautological (non-contradictory) and $\forall$-reduced ($\exists$-reduced).

The literal $\ell_\exists$ is an *existential unit* in clause C iff $\mathcal{R}_\forall^{\mathcal{Q}}(C) = \{\ell_\exists\}$. Then C is called a *unit clause*. Unit cubes and universal units are defined analogously. We also simply call C a unit, if C is a unit clause or unit cube. We further extend these definitions to clauses C (and cubes) under an assignment A, by using $C[A]$ instead of C. For instance, $\ell_\exists$ is an existential unit in C under A iff $\ell_\exists$ is an existential unit in $C[A]$. In this case we assume that C is not satisfied by A, i.e., $C[A] \neq \top$. A literal is called pure in QBF $\mathcal{Q}(\varphi)$ if it occurs only in exactly one polarity.

2.3 Abstract QCDCL Solving

Our abstract QCDCL solver is described in terms of a state transition system. It explicitly traverses the assignment tree of a QBF given in CNF and DNF to prove its (un)satisfiability. States S of the form $A \parallel \mathcal{D} \parallel \mathcal{C}$ consist of a CNF $\mathcal{C}$, a DNF $\mathcal{D}$, and of an assignment A (also called *trail* in solver implementations) over variables of the fixed quantifier prefix $\mathcal{Q}$. The literals in A are either decided or implied (see below). A decision literal is written as ℓ^d. The initial state of our abstract QCDCL solver is $\emptyset \parallel \mathcal{D} \parallel \mathcal{C}$, where $\mathcal{D}$ contains the input QBF in DNF, while $\mathcal{C}$ is the input QBF in CNF. Therefore the duality property holds, i.e., $\mathcal{Q}(\mathcal{C}) \equiv \mathcal{Q}(\mathcal{D})$. Next, we introduce the rules of our abstract QCDCL solver.

Unit propagation extends the current assignment by unit literals with respect to their quantifier type.

Due to the duality property, we can identify *pure literals* either from $\mathcal{C}$ or $\mathcal{D}$. Note that this is not possible in decision procedures without duality-awareness because there $\mathcal{D}$ is not a complete representation of the input QBF.

$$\textsc{Unit}_{\exists}\colon \frac{A \parallel \mathcal{D} \parallel \mathcal{C} \wedge C}{A\, \ell_{\exists} \parallel \mathcal{D} \parallel \mathcal{C} \wedge C} \quad \ell_{\exists} \text{ existential unit in } C[A]$$

$$\textsc{Unit}_{\forall}\colon \frac{A \parallel \mathcal{D} \vee C \parallel \mathcal{C}}{A\, \neg\ell_{\forall} \parallel \mathcal{D} \vee C \parallel \mathcal{C}} \quad \ell_{\forall} \text{ universal unit in } C[A]$$

$$\textsc{Pure}_{\exists}\colon \frac{A \parallel \mathcal{D} \parallel \mathcal{C}}{A\, \ell_{\exists} \parallel \mathcal{D} \parallel \mathcal{C}} \quad \ell_{\exists} \in \mathcal{R}^{\mathcal{Q}}_{\exists}(\mathcal{D}[A]) \text{ is pure}$$

$$\textsc{Pure}_{\forall}\colon \frac{A \parallel \mathcal{D} \parallel \mathcal{C}}{A\, \neg\ell_{\forall} \parallel \mathcal{D} \parallel \mathcal{C}} \quad \ell_{\forall} \in \mathcal{R}^{\mathcal{Q}}_{\forall}(\mathcal{C}[A]) \text{ is pure}$$

$$\textsc{Decide}\colon \frac{A \parallel \mathcal{D} \parallel \mathcal{C}}{A\, \ell^{d} \parallel \mathcal{D} \parallel \mathcal{C}} \quad \ell \text{ is unassigned and all } \ell' \text{ with } \ell' <_{\mathcal{Q}} \ell \text{ are assigned in } A$$

Rule Decide adds the decision literals to the current assignment A. The quantifier prefix restricts the set of decision candidate variables. Further, each decision must preserve consistency with A.

$$\textsc{Learn}_{\text{CNF}}\colon \frac{A \parallel \mathcal{D} \parallel \mathcal{C}}{A \parallel \mathcal{D} \parallel \mathcal{C} \wedge C} \quad \mathcal{C} \vDash_{\mathcal{Q}} C$$

$$\textsc{Learn}_{\text{DNF}}\colon \frac{A \parallel \mathcal{D} \parallel \mathcal{C}}{A \parallel \mathcal{D} \vee C \parallel \mathcal{C}} \quad \mathcal{D} \vDash^{\mathcal{Q}} C$$

$\textsc{Learn}_{\text{CNF}}$ and $\textsc{Learn}_{\text{DNF}}$ describe the clause and cube learning of the solver. The only restriction on learned clauses, and dually for cubes, is that they are implied by the formula w.r.t. the prefix, which by definition requires $\mathcal{Q}(\mathcal{C}) \equiv \mathcal{Q}(\mathcal{C} \wedge C)$. Deciding equivalence of two QBFs is PSPACE hard. In practice polynomial derivation techniques are used, e.g., some form of Q-resolution.

$$\textsc{Undo}_{\exists}\colon \frac{A\, \ell^{d}_{\exists} A' \parallel \mathcal{D} \parallel \mathcal{C}}{A \parallel \mathcal{D} \parallel \mathcal{C}}$$

$$\textsc{Undo}_{\forall}\colon \frac{A\, \ell^{d}_{\forall} A' \parallel \mathcal{D} \parallel \mathcal{C}}{A \parallel \mathcal{D} \parallel \mathcal{C}}$$

The Undo-rules are responsible for backtracking. These steps have no side condition, but at least one decision literal in the current assignment is required. Backtracking is allowed only precisely before such a decision literal, because backtracking to other points of the trail could lead to unnecessary repetitions of steps.

$$\textsc{Final}_{\text{CNF}}\colon \frac{A \parallel D \parallel \mathcal{C} \wedge \emptyset}{\bot}$$

$$\textsc{Final}_{\text{DNF}}\colon \frac{A \parallel \mathcal{D} \vee \emptyset \parallel \mathcal{C}}{\top}$$

If the empty clause (cube) is in $\mathcal{C}$ ($\mathcal{D}$), the formula simplifies to $\bot$ ($\top$). We also denote such a state by $\bot$ ($\top$). If the application of a rule causes a transition from a state $S = (A \parallel \mathcal{D} \parallel \mathcal{C})$ to a state $S' = (A' \parallel \mathcal{D}' \parallel \mathcal{C}')$, we denote this by $S \vdash S'$. For multiple rule applications we write $S \vdash^* S'$. Now we obtain the following lemmas from our definitions and, for instance, using facts from [65].

Lemma 2.3.1. *If $S \vdash S'$ using the* Learn *or* Final *rules, then $\mathcal{Q}(\mathcal{C}) \equiv \mathcal{Q}(\mathcal{C}')$ and $\mathcal{Q}(\mathcal{D}) \equiv \mathcal{Q}(\mathcal{D}')$. For all other rules, $\mathcal{C}' = \mathcal{C}$ and $\mathcal{D}' = \mathcal{D}$.*

Lemma 2.3.2. *If $S \vdash S'$ using rules* Unit *or* Pure*, then $\mathcal{Q}(\mathcal{C}[A]) \equiv \mathcal{Q}(\mathcal{C}[A'])$ and $\mathcal{Q}(\mathcal{D}[A]) \equiv \mathcal{Q}(\mathcal{D}[A'])$.*

Corollary 2.3.3. *Each rule preserves the duality property.*

To get closer to real QBF solvers and to enforce termination, we have to restrict the application of rules based on the actual state of the solver.

Definition 2.3.1 (Leaf Condition)**.** *State $A \parallel \mathcal{D} \parallel \mathcal{C}$ has*

$$\begin{aligned} \textit{(Conflict Condition) } L_\bot & \quad \textit{iff } \emptyset \in \mathcal{R}^{\mathcal{Q}}_\forall(\mathcal{C}[A]) \\ \textit{(Satisfaction Condition) } L_\top & \quad \textit{iff } \emptyset \in \mathcal{R}^{\mathcal{Q}}_\exists(\mathcal{D}[A]) \\ \textit{(Leaf Condition) } L & \quad \textit{iff } L_\top \vee L_\bot \end{aligned}$$

The duality property guarantees $L_\bot$ and $L_\top$ to be mutually exclusive. Whenever the solver finds a satisfying or falsifying assignment, we say that it reached a leaf of the assignment tree. Then L is true in that state.

Definition 2.3.2. *A state has the* Propagation Condition*, denoted P, if one of the* Unit *or* Pure *rules can be applied.*

Definition 2.3.3 (Driving Condition)**.** *In $AA' \parallel \mathcal{D} \parallel \mathcal{C}$ assume A' contains a decision ℓ^d, where ℓ^d is existential if C is a clause in $\mathcal{C}$ and universal if C is a cube in $\mathcal{D}$.*

$$\begin{aligned} \textit{(Driving Clause) } D_\bot(C) & \quad \textit{iff } C[AA'] = \emptyset,\ C[A] \textit{ unit} \\ \textit{(Driving Cube) } D_\top(C) & \quad \textit{iff } C[AA'] = \emptyset,\ C[A] \textit{ unit} \\ \textit{(Driving Condition) } D(C) & \quad \textit{iff } D_\bot(C) \vee D_\top(C) \end{aligned}$$

The driving condition holds (i.e., D is true) in a state $AA' \parallel \mathcal{D} \parallel \mathcal{C}$ iff there is a clause (cube) C in $\mathcal{C}$ ($\mathcal{D}$) s.t. $D(C)$ holds. Then C is a driving clause (cube) *and C is driving the existential (universal) unit literal $\ell' \in C[A]$.*

If a driving clause C is learned in state $AA' \parallel \mathcal{D} \parallel \mathcal{C}$, then the Undo rule can be applied to remove A' from the trail and then the Unit rule can be used to add $C[A]$ (the driven literal) to the current assignment. In that way, backjumping can be simulated with a sequence of small steps. This makes the whole process more transparent.

Table 2.1: Additional strategy constraints.

Rule	Pre-condition	Post-condition
UNIT	$\neg F \wedge \neg L \wedge \neg D$	
PURE	$\neg F \wedge \neg L \wedge \neg D$	
DECIDE	$\neg F \wedge \neg L \wedge \neg D \wedge \neg P$	
LEARN	$\neg F \wedge \quad L \wedge \neg D$	$D \vee F$
UNDO	$\neg F \wedge \quad L \wedge \quad D$	$\neg D$
FINAL	F	

Definition 2.3.4 (Final Condition)**.** *In state $A \parallel \mathcal{D} \parallel \mathcal{C}$ we define the following final conditions:*

$$\begin{aligned} \textit{(Inconsistency Condition)}\ F_\bot \quad & \textit{iff } \emptyset \in \mathcal{C} \\ \textit{(Tautology Condition)}\ F_\top \quad & \textit{iff } \emptyset \in \mathcal{D} \\ \textit{(Final Condition)}\ F \quad & \textit{iff } F_\top \vee F_\bot \end{aligned}$$

Lemma 2.3.4. *If in $A \parallel \mathcal{D} \parallel \mathcal{C}$, F or D holds, then also L.*

Now, to guarantee termination, a strategy for applying our rules is enforced by further restricting their side conditions (see Table 2.1). Our strategy requires to stop propagation as soon as a leaf is reached (which in turn is required for learning to be applied). In SAT it has been considered to lift this requirement, which however requires additional care during learning [111]. It is unclear at this point whether this also applies to our calculus or QBF. Decisions can be made (rule DECIDE) only when no other rule is applicable but the Final Condition does not hold yet. The strongest constraint is introduced for the LEARN rules. It ensures that learning prunes the search space. Hence, the learned clause (or cube) is either empty or driving. The UNDO-rules force the solver to backtrack exactly where the driving clause (cube) leads. Below we assume the constraints of Table 2.1.

Lemma 2.3.5. *If in $A \parallel \mathcal{D} \parallel \mathcal{C}$ there is **no** existential (universal) decision literal in A and $\neg F \wedge \neg D$ and $L_\bot$ ($L_\top$) hold, then $\mathcal{C} \models_Q \emptyset$ ($\mathcal{D} \models^Q \emptyset$).*

Proof. By semantics of QBF and Lemma 2.3.2. □

Lemma 2.3.6. *If in $A \parallel \mathcal{D} \parallel \mathcal{C}$ there is an existential (universal) decision literal in A and $\neg F \wedge \neg D$ and $L_\bot$ ($L_\top$) hold, then there exists a driving clause (cube) C s.t. $\mathcal{C} \models_Q C$ ($\mathcal{D} \models^Q C$).*

Proof. Given $A \parallel \mathcal{D} \parallel \mathcal{C}$ where $\neg F \wedge L_\bot \wedge \neg D$ holds, i.e., $\emptyset \notin \mathcal{C}$, but $\emptyset \in \mathcal{R}_\forall^Q(\mathcal{C}[A])$ and there is no driving clause in $\mathcal{C}$. Then A has the form $L_0 \ell_1^d L_1 ... \ell_n^d L_n$ for some $n > 0$, where $\ell_1^d, ..., \ell_n^d$ are the existential decision literals of A and $L_0, ..., L_n$ contain the implied literals (from UNIT and PURE rules) and the universal decision literals. Let C' be the clause $(\neg\ell_1^d \vee ... \vee \neg\ell_n^d)$. Then by construction $C'[A] = \emptyset$ and

$\neg \ell_n^d$ is an existential unit in C' under the assignment $A' = \{L_0 \ell_1^d L_1 ... \ell_{n-1}^d L_{n-1}\}$ (which is a prefix of A), thus C' is driving ℓ_n^d. We further have to show that $\mathcal{C} \vDash_{\mathcal{Q}} C'$, i.e. $\mathcal{Q}(\mathcal{C}) \equiv \mathcal{Q}(\mathcal{C} \wedge C')$. Due to Lemma 2.3.2 and QBF semantics, if $\mathcal{Q}(\mathcal{C})$ is true, then there is no tree model with a branch that contains $\ell_1^d, ..., \ell_n^d$. Therefore, conjoining $(\neg \ell_1^d \vee ... \vee \neg \ell_n^d)$ to $\mathcal{Q}(\mathcal{C})$ is satisfiability preserving. The case $L_\top$ is analogous. □

In practice the learned clause or cube is not built as in the proof above. Conflict and solution analysis of QCDCL solvers rely on some form of Q-resolution, where the derived clause or cube can be safely added to the formula by construction. We further obtain the following facts, without complete proofs, due to space constraints.

Lemma 2.3.7. *There are no infinite derivations of the form* $(\emptyset \parallel \mathcal{D} \parallel \mathcal{C}) \vdash S_1 \vdash S_2 \vdash \cdots \vdash S_i \vdash \cdots$

Lemma 2.3.8. *If* $(\emptyset \parallel \mathcal{D} \parallel \mathcal{C}) \vdash^* S$ *and no rule applies to* S *then* S *is either* $\bot$ *or* $\top$.

Lemma 2.3.9. *If* $(\emptyset \parallel \mathcal{D} \parallel \mathcal{C}) \vdash^* S \in \{\bot, \top\}$, $Q(\mathcal{C}) \equiv S$.

Theorem 2.3.10. *Our abstract QCDCL calculus is sound and complete. Applying the additional strategy constraints always produces terminating derivations.*

2.4 Extensions

In our framework termination depends on learning of the empty clause or cube. Thus, in its basic form, it can not simulate pure search-based solvers without learning. Further, as memory is limited in practice, it is necessary to forget learned clauses and cubes which became irrelevant. Moreover, in practice, it can be beneficial to stop the current search and start over again with an empty trail. Extending the framework with further rules, we can easily capture also these aspects of practical solvers.

$$\textsc{Forget}_{\text{CNF}}\colon \frac{A \parallel \mathcal{D} \parallel \mathcal{C} \wedge C}{A \parallel \mathcal{D} \parallel \mathcal{C}} \quad \mathcal{C} \vDash_{\mathcal{Q}} C$$

$$\textsc{Forget}_{\text{DNF}}\colon \frac{A \parallel \mathcal{D} \vee C \parallel \mathcal{C}}{A \parallel \mathcal{D} \parallel \mathcal{C}} \quad \mathcal{D} \vDash^{\mathcal{Q}} C$$

The side conditions of the Forget-rules guarantee that only redundant information is discarded. With these additional rules we can also simulate solvers without learning, if the strategy enforces to apply Forget right after backtracking (Undo) and propagation (Unit & Pure). Obviously, as soon as an empty clause or cube is learned, Final termination rules have to be applied immediately. Restart of the search has no side condition, but on the strategy level additional care is necessary in order to maintain termination.

$$\textsc{Restart}: \frac{A \parallel \mathcal{D} \parallel \mathcal{C}}{\emptyset \parallel \mathcal{D} \parallel \mathcal{C}}$$

$$\textsc{Learn'}_{\text{DNF}}: \frac{A \parallel \mathcal{D} \parallel \mathcal{C}}{A \parallel \mathcal{D} \vee C \parallel \mathcal{C}} \quad \mathcal{D} \models^{\mathcal{Q}} C \text{ or } C \models \mathcal{C}$$

Usually the input of a QBF solver is only available in PCNF, therefore we can not assume the duality property as invariant. However, the following weaker invariant over the clauses and cubes serves a similar purpose: $\mathcal{Q}(\mathcal{D}) \Rightarrow \mathcal{Q}(\mathcal{C})$. This invariant ensures that whenever the DNF is satisfied, the CNF is satisfied as well. To adapt our framework to this new invariant, some modifications are necessary. For instance, $\textsc{Pure}_{\exists}$ has to search for the existential pure literals in the clause set (instead of the cube set). Moreover, since in that case the DNF is incomplete, $\textsc{Forget}_{\text{DNF}}$ can be applied without side condition, and the constraints of learning new cubes has to be weakened (see $\textsc{Learn'}_{\text{DNF}}$). There are now two possible solution scenarios (as in [239]). First, one of the cubes in the database is satisfied. In that case we learn as we did before and the driving cube construction remains the same. Second, all the clauses in the clause database are satisfied but no satisfied cube exists. Note that the Satisfaction Condition in Def. 2.3.1 has to be updated.

Definition 2.4.1 (Satisfaction Condition)**.** *$A \parallel \mathcal{D} \parallel \mathcal{C}$ has*

(DNF Satisfaction) S_{DNF}	*iff* $\emptyset \in \mathcal{R}_{\exists}^{\mathcal{Q}}(\mathcal{D}[A])$
(CNF Satisfaction) S_{CNF}	*iff* $\mathcal{C}[A] = \top$
(Satisfaction Cond.) $L_{\top}$	*iff* $S_{\text{CNF}} \vee S_{\text{DNF}}$

Since we can learn cubes which are weakening the DNF, Lemma 2.3.1 ceases to hold. Instead we obtain:

Lemma 2.4.1. *If $S \vdash S'$ using the* Learn *or* Final *rules, then $\mathcal{Q}(\mathcal{C}) \equiv \mathcal{Q}(\mathcal{C}')$ and $\mathcal{Q}(\mathcal{D}') \Rightarrow \mathcal{Q}(\mathcal{C}')$.*

2.5 Conclusion and Future Work

We presented a formal framework for concisely capturing search-based QBF solving. Such a framework is useful for better understanding of various types of QCDCL solvers.

We plan to use our framework to close the gap between QCDCL solving and other approaches, including expansion-based techniques [120,134]. Currently, it is not clear how these approaches relate to one another w.r.t. solving strength. While there is some work on relating proof systems (e.g., [135]), the actual search strategies have not been compared yet. Especially when different techniques are integrated as currently proposed in [166], a better understanding of the individual solving techniques is indispensable.

2.6 Acknowledgment

The authors would like to thank the anonymous reviewers for their helpful comments. This research has been supported by the Austrian Science Fund (FWF) under projects W1255-N23 and S11408-N23.

2.7 Appendix

Example 2.7.1. *Consider the QBF* $\psi = \exists x \forall y.\ x \Leftrightarrow y$*. It can be transformed to CNF as* $\exists x \forall y \exists p.\ p \wedge (\neg p \vee \neg x \vee y) \wedge (\neg p \vee x \vee \neg y)$*, and to DNF as* $\exists x \forall y q.\ q \vee (\neg q \wedge \neg x \wedge \neg y) \vee (\neg q \wedge x \wedge y)$*, where* p *and* q *are newly introduced Tseitin-variables. Given these two representations of the input formula* ψ*, the initial state of the abstract solver is* $\emptyset \parallel \mathcal{D} \parallel \mathcal{C}$*, where* $\mathcal{Q} = \exists x \forall y \exists p \forall q$*,* $\mathcal{D} = q \vee (\neg q \wedge \neg x \wedge \neg y) \vee (\neg q \wedge x \wedge y)$ *and* $\mathcal{C} = p \wedge (\neg p \vee \neg x \vee y) \wedge (\neg p \vee x \vee \neg y)$*. Note that* p *and* q *are interchangeable in the prefix, thus* $\exists x \forall y q \exists p$ *is another possible prefix for that example. A possible derivation from that state would be as follows.*

$$\begin{aligned}
&\emptyset && \parallel \mathcal{D} && \parallel \mathcal{C} && \vdash_{\textsc{Unit}_\exists} && (2.1)\\
&p && \parallel \mathcal{D} && \parallel \mathcal{C} && \vdash_{\textsc{Unit}_\exists} && (2.2)\\
&p\,x && \parallel \mathcal{D} && \parallel \mathcal{C} && \vdash_{\textsc{Learn}_{\text{CNF}}} && (2.3)\\
&p\,x && \parallel \mathcal{D} && \parallel \mathcal{C} \wedge \emptyset && \vdash_{\textsc{Final}_{\text{CNF}}} && (2.4)\\
& && && \bot && && (2.5)
\end{aligned}$$

In the initial state there is an existential unit (p*) in the CNF and there is a universal unit (*q*) in the DNF formula. Assume that the solver first propagates* p*, that yields state (2.2). In that state, there is still* q *as universal unit. Further, the second and third clauses of the CNF formula are the existential units* $\neg x$ *and* x *respectively, since under the assignment* p *the universal literals* y *and* $\neg y$ *are reduced.*

Consider the case that the solver propagates x *as a next step, which yields state (2.3). Here the second clause of the CNF formula is falsified, i.e.* $\emptyset \in \mathcal{R}^{\mathcal{Q}}_{\forall}(\mathcal{C}[p, x])$*, so the Conflict Condition (*$L_\bot$*) holds. Since there is no decision literal on the trail, no driving clause can be constructed, but the empty clause can be learned. After that step, in state (2.4),* $F_\bot$ *holds, therefore the only rule that can be applied is* $\textsc{Final}_{\text{CNF}}$*, that yields the state* $\bot$*.*

It is not hard to see, that if the initial formula would have been $\forall x \exists y.\ x \Leftrightarrow y$ *(instead of* $\exists x \forall y.\ x \Leftrightarrow y$*), then the derivation of state* $\top$ *could have been the dual of the above steps (i.e. apply rules* $\textsc{Unit}_\forall$*,* $\textsc{Learn}_{\text{DNF}}$*,* $\textsc{Final}_{\text{DNF}}$ *instead of the rules* $\textsc{Unit}_\exists$*,* $\textsc{Learn}_{\text{CNF}}$*,* $\textsc{Final}_{\text{CNF}}$ *respectively).*

Example 2.7.2. *Consider the initial state of the abstract solver as* $\emptyset \parallel \mathcal{D} \parallel \mathcal{C}$*, where* $\mathcal{Q} = \forall x \exists y \forall z$*,* $\mathcal{C} = (x \vee y) \wedge (\neg x \vee \neg y)$ *and* $\mathcal{D} = (x \wedge \neg y \wedge z) \vee (x \wedge \neg y \wedge \neg z) \vee (\neg x \wedge y \wedge z) \vee (\neg x \wedge y \wedge \neg z)$*. A possible derivation from that state would*

be as follows.

$$
\begin{array}{llll}
\emptyset & \| \mathcal{D} & \| \mathcal{C} & \vdash_{\textsc{Decide}} \quad (2.6) \\
x^d & \| \mathcal{D} & \| \mathcal{C} & \vdash_{\textsc{Pure}_\exists} \quad (2.7) \\
x^d \neg y & \| \mathcal{D} & \| \mathcal{C} & \vdash_{\textsc{Unit}_\forall} \quad (2.8) \\
x^d \neg y \neg z & \| \mathcal{D} & \| \mathcal{C} & \vdash_{\textsc{Learn}_{\textsc{dnf}}} \quad (2.9) \\
x^d \neg y \neg z & \| \mathcal{D} \vee x & \| \mathcal{C} & \vdash_{\textsc{Undo}} \quad (2.10) \\
\emptyset & \| \mathcal{D} \vee x & \| \mathcal{C} & \vdash_{\textsc{Unit}_\forall} \quad (2.11) \\
\neg x & \| \mathcal{D} \vee x & \| \mathcal{C} & \vdash_{\textsc{Pure}_\exists} \quad (2.12) \\
\neg x\, y & \| \mathcal{D} \vee x & \| \mathcal{C} & \vdash_{\textsc{Unit}_\forall} \quad (2.13) \\
\neg x\, y \neg z & \| \mathcal{D} \vee x & \| \mathcal{C} & \vdash_{\textsc{Learn}_{\textsc{dnf}}} \quad (2.14) \\
\neg x\, y \neg z & \| \mathcal{D} \vee x \vee \emptyset & \| \mathcal{C} & \vdash_{\textsc{Final}_{\textsc{dnf}}} \quad (2.15) \\
 & & \top & \quad (2.16)
\end{array}
$$

Initially, the only possible step is to apply rule Decide*, since* $\neg P \wedge \neg F \wedge \neg L \wedge \neg D$ *holds. The only variable satisfying the side condition of rule* Decide *is* x*. Assume it is decided to be true. Then in state (2.7) there is* $\neg y$ *as existential unit in* $\mathcal{R}^{\mathcal{Q}}_{\forall}(\mathcal{C}[x])$ *and at the same time it is pure in* $\mathcal{R}^{\mathcal{Q}}_{\exists}(\mathcal{D}[x])$*. Assume that pure literal propagation has higher priority and the solver propagates then* $\neg y$ *using rule* $\textsc{Pure}_\exists$*.*

In state (2.8), although all clauses are satisfied in $\mathcal{C}$ *by the current assignment, there is neither an empty clause nor empty cube in* $\mathcal{C}$ *or* $\mathcal{D}$*, so the Leaf Condition does not hold. But, there are* z *and* $\neg z$ *as universal units in cubes* $(x \wedge \neg y \wedge z)$ *and* $(x \wedge \neg y \wedge \neg z)$ *respectively. Assume that the solver decides to use the first cube for propagation, which extends the current assignment with* $\neg z$*. With this step (in state (2.9)) the second cube becomes empty. Therefore* $L_\top$ *(and thus* L*) holds. Since there is no driving cube in the DNF (otherwise in the very first step unit propagation instead of decision would have been possible), the only rule that is applicable is* $\textsc{Learn}_{\textsc{dnf}}$*. The solver learns the driving cube* x*, which yields state (2.10). Now there is a driving cube, so rule* Undo *is applicable to backtrack. In state (2.11) the recently learned cube becomes universal unit, therefore* $\textsc{Unit}_\forall$ *is a valid step and extends the current assignment with* $\neg x$*. Then, there is* y *as existential unit in* $\mathcal{R}^{\mathcal{Q}}_{\forall}(\mathcal{C}[\neg x])$ *and as pure in* $\mathcal{R}^{\mathcal{Q}}_{\exists}(\mathcal{D}[\neg x])$*. Application of rule* $\textsc{Pure}_\exists$ *yields state (2.13). Just like in state (2.7),* z *and* $\neg z$ *are universal units, but this time in the cubes* $(\neg x \wedge y \wedge z)$ *and* $(\neg x \wedge y \wedge \neg z)$ *respectively. After unit propagation, there is an empty cube in* $\mathcal{R}^{\mathcal{Q}}_{\exists}(\mathcal{D}[\neg x, y, \neg z])$*, so* $L_\top$ *holds. This time there is no universal decision literal on the trail, thus the only cube to learn is* $\emptyset$*. Then* $F_\top$ *(and so* F*) holds, thus, finally, the only allowed step is* $\textsc{Final}_{\textsc{dnf}}$*, which terminates the derivation.*

Chapter 3

Implicit Hitting Set Algorithms for Maximum Satisfiability Modulo Theories

Published

In Proceedings of the 9th International Joint Conference on Automated Reasoning (IJCAR 2018), pages 134-151, held as Part of the Federated Logic Conference, FloC Oxford, UK. See [93].

Authors

Katalin Fazekas, Fahiem Bacchus and Armin Biere.

Abstract

Solving optimization problems with SAT has a long tradition in the form of MaxSAT, which maximizes the weight of satisfied clauses in a propositional formula. The extension to maximum satisfiability modulo theories (MaxSMT) is less mature but allows problems to be formulated in a higher-level language closer to actual applications. In this paper we describe a new approach for solving MaxSMT based on lifting one of the currently most successful approaches for MaxSAT, the implicit hitting set approach, from the propositional level to SMT. We also provide a unifying view of how optimization, propositional reasoning, and theory reasoning can be combined in a MaxSMT solver. This leads to a generic framework that can be instantiated in different ways, subsuming existing work and supporting new approaches. Experiments with two instantiations clearly show the benefit of our generic framework.

3.1 Introduction

SMT solvers have become indispensable tools for solving a wide range of problems in many areas. Such solvers provide either a satisfying assignment (e.g., a witness for a bug) or a proof of unsatisfiability (e.g., proving that a particular abstraction does not display a bug). However, in many applications the problem to be solved is more naturally cast as an optimization problem: find an assignment that minimizes some cost function. Li et al. [159], for instance, give a range of applications where optimization is critical. The need to solve such applications has led to a range of work addressing optimization in SMT (e.g., [45, 74, 159, 171, 192, 216, 217, 218]).

Work on SMT optimization varies in the generality of the objective functions that can be modeled. For example, [159, 216] address optimizing objective functions stated in the theory of linear real arithmetic, while [171] can deal with linear objective functions in which some variables are restricted to be integer. MaxSMT [192] is a restricted but important sub-problem in which the objective functions are linear expressions over Boolean variables (Pseudo Boolean expressions).

In this paper we focus on MaxSMT. Although MaxSMT is not as general as some other optimization approaches, MaxSMT specific solvers are often more efficient on problems where Pseudo Boolean objectives suffice [218], and recent rapid progress in the efficiency of MaxSAT solvers [5] indicates that this special case may more likely scale to practical problems than more general optimization approaches. Furthermore, MaxSAT already has a wide and growing range of applications including planning, fault localization in C code, design debugging, and a variety of problems in data analysis (see [15]). This indicates that Pseudo Boolean objectives are sufficient in a range of applications, and hence MaxSMT, with its addition of theories, is likely to have even greater applicability.

The implicit hitting set (IHS) approach [79] for solving MaxSAT has seen considerable recent progress and is now one of the most effective ways of solving MaxSAT. For example, IHS solvers have been the top performing solvers on weighted problems in the most recent 2016 and 2017 evaluations of MaxSAT solvers [5]. One of the key benefits of the IHS approach is that it provides a clear separation between optimization and propositional reasoning. In particular, in IHS solvers optimization is performed by a separate minimum cost hitting set solver, while the SAT solver is used solely for propositional reasoning. This separation of concerns supports the observed improved performance by allowing the exploitation of more efficient specialized solvers for each component.

Since MaxSAT and MaxSMT are quite similar problems, this naturally leads to the question of how MaxSMT can be similarly separated into optimization, propositional reasoning, and theory reasoning. In this paper we provide a general view of how these separate components can be combined to solve MaxSMT by providing a formal reasoning calculus [194] for MaxSMT solvers that achieves a clear separation of these different components. The calculus formalizes a notion

of state that abstracts the more complex notions of state used in implemented solvers, and a set of inference rules for transforming the state that abstracts the operations performed by implemented solvers. The power of the calculus is that almost any scheme for scheduling the application of these rules leads to a solution. Hence, it supports the design of a wide range of different implementations of the basic inferences and of control structures for scheduling their application. It also provides a formal framework for effective harvesting of advances in MaxSAT for improving MaxSMT and vice versa.

3.2 Preliminaries

We consider formulas F in conjunctive normal form (CNF) consisting of a set of clauses, where each clause C is a disjunction of literals, which are first-order atoms or propositional variables, or their negation. MaxSAT problems are specified by a purely propositional CNF F, without first-order atoms, partitioned into **hard** and **soft** clauses, $\mathit{hard}(F)$ and $\mathit{soft}(F)$. A **feasible solution** to the MaxSAT problem is a truth assignment that satisfies all of the hard clauses. A **core** in MaxSAT solving is a *set of soft clauses* (a subset of $\mathit{soft}(F)$) that when combined with the hard clauses forms an unsatisfiable set of clauses.

Each soft clause C has a positive weight, denoted by $\mathit{cost}(C)$, which specifies the cost of falsifying it. The cost of a set of soft clauses S is the sum of the costs of the soft clauses it contains: $\mathit{cost}(S) = \sum_{C \in S} \mathit{cost}(C)$. The cost of a feasible solution π is the sum of the costs of the soft clauses it falsifies: $\mathit{cost}(\pi) = \mathit{cost}(\{C \in S \mid \pi \not\models C\})$. An **optimal solution** for a MaxSAT problem is a feasible solution with *minimum cost* among all feasible solutions. Solving a MaxSAT problem is the task of finding an optimal solution.

We can restrict our attention, w.l.o.g., to formulas F in which all soft clauses are unit. In particular, any non-unit soft clause C can be converted to a unit soft clause by (i) adding a new (*relaxed*) hard clause $C \vee v$ where v is a *new* propositional variable (called a *relaxing* or *selector* variable), and (ii) replacing the soft clause C with a new unit soft clause $(\neg v)$ with $\mathit{cost}((\neg v)) = \mathit{cost}(C)$. This transformation is sound since any optimal solution satisfies $C \leftrightarrow \neg v$.

Considering ground first-order atoms generalizes MaxSAT to MaxSMT [192], as SMT [215] generalizes SAT. As in MaxSAT, a MaxSMT problem consists of a set of hard and soft clauses with each soft clause having a weight. However, in MaxSMT literals can be formed from theory atoms as well as from propositional variables. For example, over the theory of linear real arithmetic (LRA) we could form clauses like $(p \vee \neg(1 \leq y) \vee (x + y \geq 2))$, with a propositional variable p and LRA theory atoms $(1 \leq y)$ and $(x + y \geq 2)$.

Let $\mathit{atoms}(F)$ be the set of atoms in F, which range over propositional variables as well as theory atoms. We extend this notion to literals, clauses, and sequences of literals accordingly. A (partial truth) **assignment** over $\mathit{atoms}(F)$ is a sequence of literals from $\mathit{atoms}(F)$ that (i) does not contain both x and $\neg x$

for any $x \in atoms(F)$ and (ii) has no repeated literals. If A and M are two sequences of literals we write AM to indicate their concatenation. An assignment π over $atoms(F)$ is called a **propositional model** of F, denoted by $\pi \models F$, if it satisfies the Boolean abstraction of F in which theory atoms are treated simply as new independent propositional variables. A propositional model of F, π, is also a **theory consistent model** of F if the conjunction of theory literals made true by π is consistent with all theory axioms, denoted $\pi \models_T F$.

A feasible solution π for a MaxSMT formula F is required to be theory consistent model of $hard(F)$ ($\pi \models_T hard(F)$). The cost of π is defined as in MaxSAT. Accordingly, solving MaxSMT means finding an optimal (minimum cost) feasible solution. Again, w.l.o.g., we can assume that all soft clauses are unit clauses. Similarly, a **core** in MaxSMT is a subset of *soft clauses* that when combined with the hard clauses does not have a theory consistent model.

Let K be a set of cores, i.e., a set of sets of soft clauses. A **hitting set** η of K is a set of soft clauses that has a non-empty intersection with every set in K: $\forall \kappa \in K. \eta \cap \kappa \neq \emptyset$. As defined above $cost(\eta) = \sum_{C \in \eta} cost(C)$.

3.3 Abstract Hitting Set based MaxSMT Solving

The main contribution of our paper is to introduce and formalize a calculus for the implicit hitting set (IHS) approach for MaxSMT, which at the same time provides the first formal calculus for the IHS approach to MaxSAT [79]. Our calculus captures a flexible separation between optimization, propositional reasoning, and theory reasoning, supporting a number of different implementation strategies. The separation between optimization and propositional reasoning is achieved by exploiting the IHS approach for solving MaxSAT/MaxSMT. Other approaches to MaxSAT solving, e.g., [6, 177, 187], employ exclusively propositional reasoning, doing optimization by solving a sequence of SAT decision problems.

Our calculus can be modified to model such approaches by combining the optimization and propositional reasoning components into a single "MaxSAT" component. This would provide a formal model of MaxSMT approaches like [74]. However, as we will demonstrate below, even without such a formal model our calculus still provides a framework for understanding the approach of [74].

IHS and the above cited approaches to solving MaxSAT use propositional reasoning to find cores by exploiting SAT solving under assumptions [88]. In particular, for any subset of soft clauses S we can determine if S and the hard clauses are satisfiable by assuming that the literal of each (unit) soft clause in S is true. If the conjunction of these literals with the hard clauses is unsatisfiable, the SAT solver assumption mechanism returns a clause falsified by the assumptions. Hence, this clause contains only negations of assumed literals, identifying a subset of S that, with the hard clauses, is unsatisfiable (i.e., a core).

Hence, as a first step towards a formal calculus for IHS MaxSMT solving, we

Table 3.1: Transition rules for solving SAT under assumptions (**A-Sat**)

UnitProp $A \mid M \mid F \Longrightarrow A \mid M\,\ell \mid F$	**if**	There is a clause $(C \vee \ell) \in F$ s.t. $AM \models \neg C$ and $atom(\ell) \notin atoms(AM)$
Decide $A \mid M \mid F \Longrightarrow A \mid M\,\ell^d \mid F$	**if**	$atom(\ell) \in \big(atoms(F) \setminus atoms(AM)\big)$
Backjump $A \mid M\ell^d N \mid F \Longrightarrow A \mid M\,\ell' \mid F$	**if**	There is a clause $C \in F$ s.t. $AM\ell^d N \models \neg C$ and a clause $C' \vee \ell'$ s.t. $F \models C' \vee \ell'$, $AM \models \neg C'$ and $atom(\ell') \in atoms(\ell^d N)$
Learn $A \mid M \mid F \Longrightarrow A \mid M \mid F, C$	**if**	$F \models C$ and $C \notin F$ $atoms(C) \subseteq \big(atoms(F) \cup atoms(AM)\big)$
Forget $A \mid M \mid F, C \Longrightarrow A \mid M \mid F$	**if**	$F \models C$
Sat-Model $A \mid M \mid F \Longrightarrow SAT(AM, F)$	**if**	$AM \models F$
Unsat $A \mid M \mid F \Longrightarrow conflict(F, C)$	**if**	There is a clause $D \in F$ s.t. $AM \models \neg D$ M contains no decision literals and C is a clause s.t. $F \models C$ and $A \models \neg C$

provide a calculus for assumption based SAT and SMT reasoning. To the best of our knowledge, such a calculus has not been specified before. This contribution should be useful independent of MaxSMT since assumption based reasoning is used in many different applications besides optimization.

3.3.1 SAT/SMT Solving under Assumptions

To formalize assumption based incremental SAT solving [88] and lift it to SMT, we extend the DPLL(T) calculus originally presented in [194]. As above let F be a first-order quantifier-free CNF formula over theory T. The states of our calculus are specified by a triple $A \mid M \mid F$, where F is a CNF formula (initially the input formula), A and M are non-overlapping assignments over $atoms(F)$. A is the given set of assumptions, and M is the solver's current set of implied and decided (noted by a superscript d, e.g., ℓ^d) literals.

The transition rules given in Table 3.1 specify an abstract assumption based SAT solver (**A-Sat**). These rules follow [194] but are adapted to handle assumptions; the main changes are as follows. First, the abstract states and rules have been extended with a (possibly empty) set of assumption literals A over $atoms(F)$. For example, Learn is the same, but UnitProp requires $AM \models \neg C$, instead of $M \models \neg C$. Second, we modified the rule Fail to obtain a new rule Unsat that transitions into a *conflict(F,C)* state when M has no decision literals and $AM \models \neg D$ for some $D \in F$. In that case, $F \wedge A$ must be unsatisfiable, and we can always find a clause C implied by F and falsified by A (e.g., by resolving all literals negated by M from the clause D). And third, we introduce a transition rule that leads to an explicit $SAT(AM, F)$ state when $AM \models F$ holds. This facilitates combining the assumption based transitions with a MaxSAT or MaxSMT transition system. It can be noted that our calculus captures the technique of [88] which uses one particular control scheme to derive the clauses D and C used in the Unsat rule (it is irrelevant that [88] intermixes A and M).

Table 3.2: Additional rules for solving SMT under assumptions (**A-Smt**)

T-Backjump $A \mid M\ell^d N \mid F \Longrightarrow A \mid M\,\ell' \mid F$	**if**	There is a clause $C \in F$ s.t. $AM\ell^d N \models \neg C$ and a clause $C' \vee \ell'$ s.t. $F \models_T C' \vee \ell'$, $AM \models \neg C'$ and $atom(\ell') \in atoms(\ell^d N)$
T-Learn $A \mid M \mid F \Longrightarrow A \mid M \mid F, C$	**if**	$F \models_T C$ and $C \notin F$ $atoms(C) \subseteq (atoms(F) \cup atoms(AM))$
T-Forget $A \mid M \mid F, C \Longrightarrow A \mid M \mid F$	**if**	$F \models_T C$
T-Model $A \mid M \mid F \Longrightarrow T\text{-}SAT(AM, F)$	**if**	$AM \models_T F$

Abstract assumption based SMT solving (**A-Smt**) is specified by the rules of Table 3.2 along with the rules UnitProp, Decide and Unsat of Table 3.1. Note that T-entailment subsumes propositional entailment, i.e., $F \models C$ implies $F \models_T C$. Hence, T-Learn can learn any clauses that Learn can, and T-Learn need not always employ theory reasoning (it can also use propositional reasoning to perform learning). This can be important in practice if reasoning in T is expensive. The same remark holds for T-Backjump and T-Forget.

It can also be noted that Unsat requires a falsified clause D to be in F. Hence, when $F \wedge A$ is propositionally satisfiable but T-unsatisfiable our calculus requires sufficient theory lemmas from T-Learn so as to obtain a falsified clause in F and to derive a clause C falsified by A.

We say that a state S in a transition system is **final** when no rules are applicable to it. Given a set of assumed literals A and a formula F, the initial state of assumption based SAT/SMT solving is $A \mid \emptyset \mid F$. Deciding the satisfiability/T-satisfiability of F assuming A is a derivation of the form $A \mid \emptyset \mid F \Longrightarrow \cdots \Longrightarrow S_n$, where S_n is a final state in the **A-Sat**/**A-Smt** system.

Theorem 3.3.1 (Termination)**.** *Any sequence of transitions $A \mid \emptyset \mid F \Longrightarrow \cdots$ in **A-Sat** (**A-Smt**) that contains no infinite subsequence consisting only of rules from the set* {Learn, Forget} ({T-Learn, T-Forget})*, is finite.*

Theorem 3.3.2 (Soundness)**.** *For any derivation $A \mid \emptyset \mid F \Longrightarrow \cdots \Longrightarrow S$ in **A-Sat** (**A-Smt**) where S is final w.r.t. **A-Sat** (**A-Smt**) we have*

1. *$S = conflict(F', C)$ with $F' \models C$, $A \models \neg C$ iff $F \wedge A$ is (T-)unsatisfiable.*
2. *$S = (T\text{-})SAT(AM, F')$ with $AM \models_{(T)} F'$ iff $F \wedge A$ is (T-)satisfiable.*

We can treat A as a prefix of decision literals of M that can not be changed by backjumping. Under this interpretation the results of [194] can be extended to obtain proofs for Theorems 3.3.1 and 3.3.2. We omit the details due to space constraints.

Table 3.3: Transition rules for Optimization ($*$ is any SAT/SMT state) (**A-MaxSMT**)

SAT/SMT-Transition $(LB, UB, \mu) \mid K \mid \langle * \rangle \Longrightarrow$ $\quad (LB, UB, \mu) \mid K \mid \langle *' \rangle$	**if**	$*'$ is reachable from $*$ by a single **A-Sat**/**A-Smt** transition step (see Table 3.1 and Table 3.2)
Core $(LB, UB, \mu) \mid K \mid \langle conflict(F, C) \rangle \Longrightarrow$ $\quad (LB, UB, \mu) \mid K, \kappa \mid \langle conflict(F, C) \rangle$	**if**	$\kappa = \{(\neg \ell) \mid \ell \in C\}$ and $\kappa \notin K$ (κ is set of soft clauses)
HS $(LB, UB, \mu) \mid K \mid \langle * \rangle \Longrightarrow$ $\quad (LB, UB, \mu) \mid K \mid \langle A' \mid \emptyset \mid F \rangle$	**if**	$\eta = HS(K)$ $A' = \{\ell \mid (\ell) \in (soft(F) - \eta)\}$
MinHS $(LB, UB, \mu) \mid K \mid \langle * \rangle \Longrightarrow$ $\quad (LB', UB, \mu) \mid K \mid \langle A' \mid \emptyset \mid F \rangle$	**if**	$\eta = minHS(K)$ $A' = \{\ell \mid (\ell) \in (soft(F) - \eta)\}$ $LB' = \max(LB, cost(\eta))$
ImprovedSolution $(LB, UB, \mu) \mid K \mid \langle T\text{-}SAT(AM,F) \rangle \Longrightarrow$ $\quad (LB, cost(AM), AM) \mid K \mid \langle T\text{-}SAT(AM,F) \rangle$	**if**	$cost(AM) < UB$
OptimalSolution $(LB, UB, \mu) \mid K \mid \langle * \rangle \Longrightarrow optSoln(\mu)$	**if**	$LB \geq UB$

3.3.2 IHS MaxSAT/MaxSMT Solving

To obtain an abstract IHS based MaxSMT solver we add the rules given in Table 3.3. These rules extend the states of **A-Smt** by adding K and the triple (LB, UB, μ), where K is a set of cores, LB and UB are lower and upper bounds on the cost of an optimal solution to the input CNF F, and μ is a feasible solution, represented as a sequence of literals over $atoms(F)$, with $cost(\mu) = UB$. Let **A-MaxSMT** be the transition system defined by the rules in Table 3.3 along with the rules **A-Smt**.[1] The initial state of **A-MaxSMT** is always the state $IS = (0, \infty, undef) \mid \emptyset \mid \langle \{\ell \mid (\ell) \in soft(F)\} \mid \emptyset \mid hard(F) \rangle$, i.e., we start with valid lower and upper bounds, an empty set of cores, an initial assumption that all soft clauses are satisfied, and all of the hard clauses of F.

The calculus computes a growing set of cores K, each obtained from assumption based SMT solving, and uses the two subroutines, $minHS(K)$ which returns a minimum cost hitting set of K, and $HS(K)$ which returns an arbitrary hitting set of K. It can be noted that the assumptions (initially and after the rules HS or MinHS) are always asserting that some subset of the soft clauses along with the hard clauses are satisfied. Hence, as explained above, the subset of $soft(F)$ identified by the returned conflict and added to K by rule Core must be a core. Furthermore, the assumptions always specify that all soft clauses *except* those in some hitting set η of K are true. Thus, the returned conflict must identify a *new* core κ that cannot already be in K. In particular, κ is a subset of $soft(F) - \eta$ (it is a subset of the assumed true soft clauses) but no $s \in K$ is a subset of $soft(F) - \eta$ since s contains a non-empty subset $s \cap \eta$ not in $soft(F) - \eta$.

We say that $S_1 \Longrightarrow \cdots \Longrightarrow S_n$ is a **progressing subsequence** if (a) S_1 is the result of applying the MinHS rule, (b) all transitions in the sequence arise

[1] IHS MaxSAT solvers can be obtained by using the **A-Sat** rules and replacing $T\text{-}SAT(AM, F)$ in **ImprovedSolution** with $SAT(AM, F)$.

from applying one of the **A-Smt** rules (i.e. are SAT/SMT-Transition steps), and (c) S_n is final with respect to the rules of **A-Smt** (i.e., no SAT/SMT-Transition is applicable).

Theorem 3.3.3 (Termination)**.** *If $hard(F)$ is T-satisfiable then any derivation $IS \Rightarrow S_1 \Rightarrow \cdots$ of **A-MaxSMT** is finite if it satisfies the following conditions:*

1. *contains no infinite subsequence of rules from the set* {T-Learn, T-Forget}
2. *contains no infinite subsequence not containing a progressing subsequence*
3. *always applies the transitions* OptimalSolution, ImprovedSolution *and* Core *whenever they are applicable (with* OptimalSolution *being applied first).*

Theorem 3.3.4 (Soundness)**.** *If $hard(F)$ is T-satisfiable, $IS \Rightarrow \cdots \Rightarrow S_n$ is a finite sequence of transitions in **A-MaxSMT**, and S_n is final in **A-MaxSMT**, then S_n is $optSoln(\mu)$ and μ is an optimal solution of F.*

Theorem 3.3.4 is immediate from the fact that (a) $optSoln(\mu)$ is the only final state in **A-MaxSMT**, (b) LB and UB are always valid bounds, and (c) $cost(\mu) = UB$.

Hence, the main result is that the calculus terminates under the conditions of Theorem 3.3.3. A sketch of the proof follows. First, from Theorem 3.3.1 it can be seen that all progressing subsequences must be finite, and thus any infinite sequence of transitions must contain an infinite number of progressing subsequences. Theorem 3.3.2 shows that every progressing subsequence must reach either a T-SAT or a *conflict* final state. If a *conflict* state is reached, then Core must be applied next. As explained above this must add a *new* core to K. Each core is a subset of $soft(F)$ so only a finite number of cores exist. Hence, only a finite number of progressing subsequences can end in *conflict*. Otherwise, the progressing subsequence reaches T-SAT. But this can happen only once since the feasible solution found, AM, must be an optimal solution. AM satisfies all clauses except those in a minimum cost hitting set η of K (obtained from MinHS), and hence $cost(AM) \leq cost(\eta)$. Every feasible solution π satisfies $hard(F)$ and every core is unsatisfiable when added to $hard(F)$. Hence, π must falsify at least one soft clause in every core; i.e., the set of clauses falsified by π is a hitting set of K. So by the definition of *cost* and the minimality of η, $cost(\eta) \leq cost(\pi)$, and thus $cost(AM) \leq cost(\pi)$ for every feasible solution π. Once AM is found, ImprovedSolution must be applicable and the condition $cost(AM) = UB \leq cost(\eta) \leq LB$ is achieved (MinHS ensures $cost(\eta) \leq LB$). Then OptimalSolution must be applied and the derivation terminates. In sum, under the stated conditions only a finite number of progressing subsequences can be executed and so the derivation must be finite.

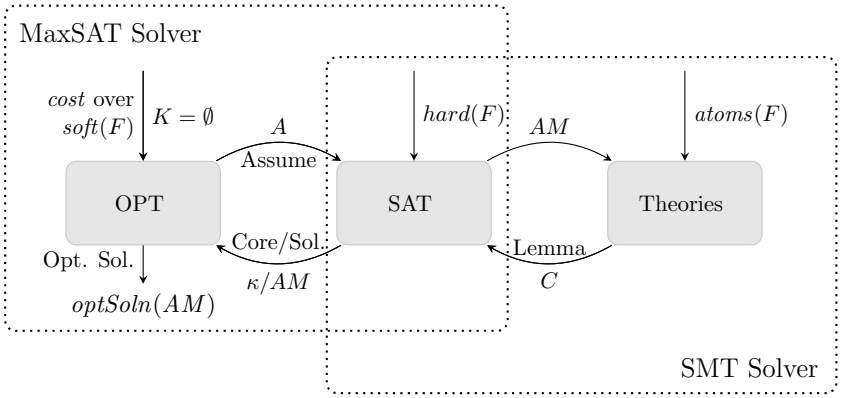


Figure 3.1: General architecture for an IHS based MaxSMT solver

3.4 Generic Hitting Set based MaxSMT

Here we present a general framework that realizes the previously introduced ideas for IHS based MaxSMT solving. Following the desiderata presented in our introduction, we decompose the problem of MaxSMT into three sub-problems: optimization (over Boolean atoms), Boolean satisfiability, and theory reasoning. Although modern SMT solvers are equipped with efficient engines for arithmetic reasoning, in MaxSMT the optimization problem depends purely on the Boolean abstraction of the formula and thus delegating the task of optimization to a specialized solver can be more efficient [218]. Figure 3.1 shows a general architecture to solve MaxSMT as an implicit hitting set problem [69, 212]. The method combines three components that are responsible for our three subtasks: OPT, an optimizer for hitting set computation; SAT, a SAT solver for Boolean reasoning; and Theories, a set of theory solvers to perform theory reasoning. The framework expects as input a MaxSMT formula (F) with a satisfiable set of hard clauses. The SAT and Theory solvers consider only the hard clauses, while the soft (unit) clauses and their costs are only considered by the optimizer. Note that we can initially check the hard clauses for satisfiability. If they are unsatisfiable there is no optimal solution.

The evaluation starts with OPT, which computes a (potentially optimal) hitting set η of the current set of unsatisfiable cores (K). This is translated into a set of assumptions (noted as A in Fig. 3.1) that requires the satisfaction of all soft clauses not in η (see HS and MinHS steps of **A-MaxSMT**).

The SAT solver can then decide if there exists a feasible solution satisfying $hard(F)$ and the assumptions. Theory solvers can be invoked at various points to check the T-consistency of the SAT solver's current partial assignment AM and to perform T-learning. As in [194] there are a range of flexible (e.g., more eager or more lazy) strategies for deciding when theory reasoning should be invoked.

For a given conjunction of theory literals, a theory solver might return a subset that is T-unsatisfiable forming a **conflict clause** after negation, or additional

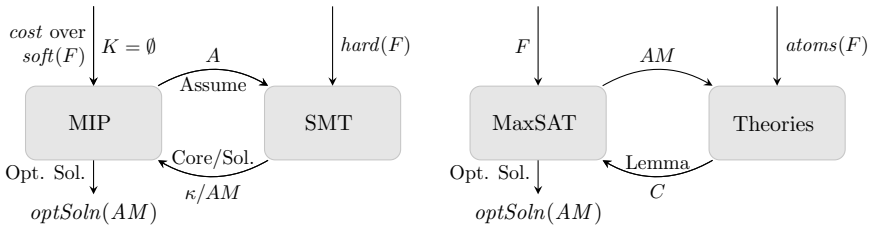


Figure 3.2: Example merged (i.e. single-SAT) instantiations of our framework

theory clauses for T-learning. In both cases the returned clauses are valid **lemmas** of the theory (C in Fig. 3.1). The SAT and theory solvers continue their collaboration under the assumption of A until either a theory consistent model of $\mathit{hard}(F) \wedge A$ is found (i.e. state $T\text{-}SAT(AM, F)$ is reached), or $\mathit{hard}(F) \wedge A$ is found to be unsatisfiable (i.e. state $\mathit{conflict}(F, C)$ is reached). In the latter case, the SAT solver constructs an unsatisfiable core (κ in Fig. 3.1) that consists of a subset of the soft clauses assumed to be satisfied in A. After that, the optimizer can compute a new hitting set that hits κ as well. Note that the new hitting set need not be of minimum cost. From the new hitting set, a new A is constructed and a new iteration starts. Any theory consistent model that is found for $\mathit{hard}(F) \wedge A$ is a feasible solution of the MaxSMT problem. The optimality of these solutions can be decided by the optimizer component based on their costs. In case the found solution is not optimal, a new hitting set is computed in order to find a better solution. Otherwise, the model is returned as a final optimal solution.

3.4.1 Possible Instantiations

A practical tool following our proposed general architecture can be achieved in various ways. Based on Fig. 3.1, one could combine a hitting set calculator with a SAT solver and a set of theory solvers. However, this implementation would not automatically benefit from the advanced techniques implemented in MaxSAT and SMT solvers nor from any future improvements to such solvers. So a more practical question is how to combine already existing tools to obtain a MaxSMT solver. Here we consider mixed integer programming solvers (MIP), for example CPLEX, for solving the minimum hitting set problem since they are widely available and display state of the art performance on a range of instances.

As Fig. 3.1 hints, some MaxSAT solvers already implement efficient collaboration between MIP and SAT solvers, while SMT solvers combine SAT and theory solvers. Combining these solvers as black-boxes results in an engine that contains two SAT solvers, while merging these engines results in a tool with a single SAT solver. Figure 3.2 shows two possible instances of the latter case. On the left, we keep an SMT solver as a black-box and combine it with a MIP solver that is responsible for the hitting sets (and so the assumptions) in each

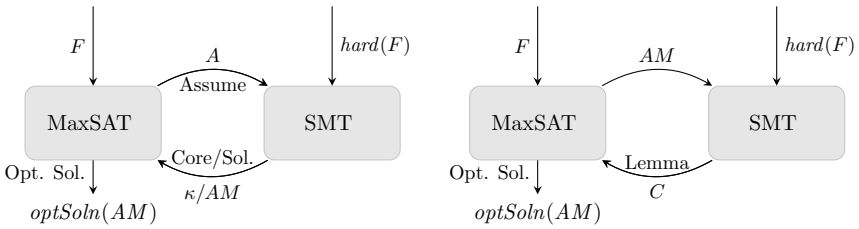


Figure 3.3: Example combined (i.e. double-SAT) instantiations of our framework

iteration. A benefit of this instance is efficient SMT solving and the ability to use the full power of the MIP solver to express complex objective functions (e.g., multi-objective optimization). One disadvantage is the lack of MaxSAT preprocessing and simplifications. A tighter combination could replace the SAT solver in an IHS based MaxSAT solver with an SMT solver. However, in IHS MaxSAT solving SAT calls are considered relatively cheap compared to MIP calls [80], but SMT calls can be more expensive, so the tradeoffs of some techniques would have to be reevaluated. The instance on the right side of Fig. 3.2 considers a (not necessarily IHS based) MaxSAT solver as a black-box to find an optimal solution for the abstraction of the problem, and forms a lazy lemmas on demand structure with a set of theory solvers for theory consistency checks. The benefit here is efficient optimization solving, but the disadvantage is delayed theory support.

Instantiations in Fig. 3.3 present possibilities for combining black-box (i.e. not necessarily IHS based) MaxSAT and SMT solvers, providing the advantage of efficient optimization and SMT solving at the same time. These combinations contain multiple SAT solvers where the connecting interface determines the work distribution among them. On the left side, the solvers communicate via assumptions and cores or solutions. Whenever the MaxSAT solver finds an optimal propositional model for its current problem, the SMT solver has to verify that the soft clauses satisfied in that model are also T-satisfiable (via a set of assumptions that forces their satisfaction). If not, it returns a new core to refine the MaxSAT problem. In practice, the effectiveness of this instance would be compromised if many iterations are needed to refine the MaxSAT model. Another possible disadvantage of this instance is that the SMT solver could learn lemmas that would be useful to the MaxSAT engine but are never passed to it.

An alternate instance (right side in Fig. 3.3) involves the MaxSAT solver giving to the SMT solver the complete propositional model it found (the optimal model for its current problem). If that model is not theory consistent the SMT solver can return any number of lemmas to refine the MaxSAT problem. In this instance the MaxSAT solver can learn theory related constraints from the SMT solver beyond unsatisfiable cores. The approach introduced in [74] can be seen as a combination of the instances in Fig. 3.3. There the MaxSAT optimal model is used to provide assumptions to the SMT solver (as in the left-hand instance),

but the SMT solver can return many lemmas to the MaxSAT solver not just cores (as in the right-hand instance). A potential drawback of that approach is that the returned lemmas might or might not be useful to the MaxSAT solver, and there is a risk of overloading the MaxSAT solver.

Based on these instances, it appears that support of assumption based incremental solving and efficient extraction of small cores are important features of the involved tools. Thus techniques that improve these aspects of solvers (e.g., [155]) have the potential to improve modular MaxSMT solvers as well. Further, note that our calculus allows the interruption of SMT calls in certain cases (see conditions in Theorem 3.3.3), which may be worth considering in practice.

3.5 Related Work

As argued in the introduction the focus of this paper is on the important class of MaxSMT solvers. Thus this section will concentrate on the closest related approaches. Additional experimental results are provided in the next section.

We modify and extend a general DPLL(T) framework introduced in [194] to formalize our MaxSMT solving approach. Another extension of DPLL(T) by Nieuwenhuis and Oliveras in [192] represents the optimization task explicitly as a set of theory constraints and progressively strengthens this theory by deriving tighter bounds. Our extension of DPLL(T) focuses only on MaxSMT problems and separates the optimization task from theory reasoning.

A modular approach was proposed by Cimatti et al. in [74] where MaxSAT and SMT solvers are employed as black-boxes for MaxSMT solving. As we showed in Section 3.4.1, our framework includes this approach. In Section 3.6 we present some empirical results comparing their approach with other instantiations.

In the context of core-guided MaxSAT solving, SMT solvers have been used instead of SAT, e.g., [7], to handle cardinality constraints more efficiently. We focus on IHS based MaxSMT solving in which no cardinality constraints are introduced into the SMT sub-problems.

Manolios et al. introduced the theoretical underpinnings of a Branch and Cut Modulo Theories framework and developed an optimization procedure where integer linear programming (ILP) and stably-infinite theories are combined [171]. Our approach delegates Boolean reasoning to a SAT solver, while in their construction this is done by the ILP solver.

3.6 Experimental Evaluation

We implemented two instantiations of our framework. Both use MathSAT5 [73] version 5.5.1 as the SMT component. Our first implementation **maxhs-msat** follows the architecture proposed on the left side of Fig. 3.3. It combines maxHS

3.0 as the optimizer with MathSAT5. To evaluate the potential of lifting theory lemmas to the MaxSAT level, as proposed in [74] and described in Section 3.4.1 as a combination of the instances in Fig. 3.3, the configuration **maxhs-msat-ll** lifts and adds all used theory lemmas to the MaxSAT solver in addition to unsatisfiable cores. Our second solver **cplex-msat** implements the architecture shown on the left of Fig. 3.2 which combines MathSAT5 directly with a hitting set solver (CPLEX 12.7 as in maxHS 3.0) as the optimizer. In this implementation the components interface only with assumptions and unsatisfiable cores.

In both solvers the optimizers compute an optimal hitting set η. In **maxhs-msat** maxHS computes an optimal solution to its current Boolean abstraction, but the clauses falsified by that solution form an optimal hitting set. The SMT solver then tests if the other soft clauses ($soft(F) - \eta$) are T-satisfiable. If not, a new core is added to the optimizer (along with additional theory lemmas in **maxhs-msat-ll**). Following [80], rather than calling the optimizer in each iteration we allow non-optimal hitting sets. In particular, the new SMT core can be added to the previous hitting set (**-djnt**), or a single minimum weight clause from the new core can be added to the hitting set (**-min**). In both cases we obtain a new (non-minimum) hitting set covering the new core. For **cplex-msat** only, we can also use CPLEX to compute a linear programming solution of the hitting set problem which when rounded up yields a new hitting set (**-lp**). In these cases we continue to use non-minimum hitting sets η' until $soft(F) - \eta'$ becomes T-satisfiable, and then we again use the optimizer to compute a hitting set with minimum cost.

We compare against two state-of-the-art MaxSMT solvers. OptiMathSAT (version 1.4.5) [217] is a general purpose Optimization Modulo Theories (OMT) solver that we use in two different configurations. The default configuration is denoted by **optimathsat-omt**, while **optimathsat-maxres** employs the maximum resolution approach of [187]. We also compare against z3 (version 4.6.0) with two different MaxSAT engine configurations (**z3-maxres** and **z3-wmax**). Note, that the hitting set based engine in z3 has been deprecated and was removed.

We considered three sets of benchmarks from three different sources. The *LL-benchmark* set consists of all 398 quantifier free MaxSMT benchmarks used in [74] with annotations replaced by soft assertions, split into 212 benchmarks over the theory of linear integer arithmetic and 186 benchmarks over linear real arithmetic. For each theory, half the instances have **U**nit weight for soft assertions, while the other half contains **R**andom weights in the interval of 1 and 100. The runtime limit on these instances was set to 20 minutes.

Our second benchmark set *LEX-benchmark*, consisting of equalities over propositional atoms, are lexicographically-optimum realization problems used in [218]. We only considered the 6098 instances where three groups of soft assertions (**T**ime, **C**ost and **W**eight) have different priorities and the objective is to lexicographically minimize the sum of the falsified assertions with respect to a given priority order of **T**, **C**, **W**). The time limit was set to 100 seconds.

Table 3.4: Results of various solvers and configurations on LL-benchmarks from [74].

Solver	LIA(212)		LRA(186)		Total	SMT%	OPT%
	U	R	U	R			
cplex-msat	82	90	85	85	342	99.22%	0.13%
cplex-msat-djnt	85	**91**	85	85	346	98.83%	0.33%
cplex-msat-min	83	86	85	85	339	99.22%	0.04%
cplex-msat-lp	84	89	85	85	343	98.26%	0.97%
maxhs-msat	85	87	85	85	342	88.15%	11.20%
maxhs-msat-djnt	86	89	85	85	345	83.85%	15.36%
maxhs-msat-min	84	89	85	85	343	92.31%	7.04%
maxhs-msat-ll	80	84	83	78	325	82.57%	15.45%
maxhs-msat-ll-djnt	78	84	83	77	322	87.97%	10.37%
maxhs-msat-ll-min	79	86	82	85	332	80.13%	17.03%
optimathsat-maxres	**87**	90	85	86	**348**	–	–
optimathsat-omt	75	72	85	85	317	–	–
z3-maxres	73	79	86	85	323	–	–
z3-wmax	69	77	**88**	**88**	322	–	–

Finally, in order to further exercise the strengths of the different approaches, we generated a set of scaled problems from one (arbitrarily chosen) QF-LIA SMT-LIB benchmark family (Bofill-scheduling waste water treatment scheduling problems from [50]). The original family contained 156 randomly generated (referred as *rand-wwtp*) and 251 industrial (*ind-wwtp*) satisfiable SMT problems. We derived instances from these SMT problems by adding randomly chosen theory atoms with random polarity as unit soft clauses. The four groups of derived instances introduced four different percentages (10%, 25%, 50% and 100%) of the atoms in the original problem as soft assertions. All instances were generated once with unit weights and once more with random weights between 1 and the total number of atoms. Due to space constraints, we only present results on instances derived from *rand-wwtp* problems, where we observed an interesting pattern. The time limit was set to 5 minutes.

The experiments were performed on a cluster in which each computing node consisted of two Intel(R) Xeon(R) E5-2620 v4 @ 2.10GHz CPUs and 128 GB of main memory. We limited memory usage of each tool to 7GB on each instance and used different time limits as described above.

Table 3.4 presents results on the LL-benchmarks. For each solver configuration the first two columns list the number of solved instances with linear integer arithmetic as background theory, where the soft assertions have **U**nit weights in the first column and **R**andom weights in the second. Analogously, the next two columns present results in linear real arithmetic. The fifth column contains the total sum of solved instances in the previous four columns. The last two columns show the percentage of time spent in the SMT and in the optimization component (considering only solved instances). The optimization component in **cplex-msat** is CPLEX, while in **maxhs-msat** it is maxHS.

It turns out that **optimathsat-maxres** outperforms the other tools and configurations on these instances, but our implementations remain competitive. Furthermore, lemma lifting (**maxhs-msat-ll** and its different configurations) reduces the percentage time spent in SMT solving, but has a negative effect

Table 3.5: Results of various solvers and configurations on LEX-benchmarks from [218].

Solver	CTW	Time[s]	WTC	Time[s]	cores	opt. HS
cplex-msat	3499	27825	2399	1942	1610031	1615150
cplex-msat-djnt	3687	5936	2399	1455	920387	137339
cplex-msat-min	3699	2479	2399	1391	909828	27245
cplex-msat-lp	3699	4564	2399	1493	1260683	19056
maxhs-msat	3699	2401	2399	1367	0	5319
maxhs-msat-djnt	3699	**2224**	2399	**1359**	0	5319
maxhs-msat-min	3699	2451	2399	1409	0	5319
maxhs-msat-ll	3699	2302	2399	1518	0	5319
maxhs-msat-ll-djnt	3699	2394	2399	1406	0	5319
maxhs-msat-ll-min	3699	2441	2399	1437	0	5319
optimathsat-maxres	3410	13851	1850	10209	–	–
optimathsat-omt	3481	9710	2068	10483	–	–
z3-maxres	3699	4555	2399	2231	–	–
z3-wmax	3651	5566	2295	9513	–	–

with respect to the number of solved instances compared to **maxhs-msat** and its different configurations. None of the involved tools appears to be sensitive to the type of weights (**U**niform vs. **R**andom). Although **cplex-msat** does not contain any MaxSAT preprocessing or simplification technique, the results of that tool in this experiment are similar to **maxhs-msat**.

Results on the LEX-benchmark are shown in Table 3.5. The 6098 problems contained two groups of problems. The first group of 3699 instances used the lexicographic preference ordering **C**ost, **T**ime and then **W**eight, and are shown in the first two columns which list the number of solved instances and the total run time used to solve them. The second group of 2399 instances used the reversed lexicographic preference and are shown in the next two columns. For our tools we also give the total number of unsatisfiable cores and of optimal hitting set calculations (considering again only solved instances) in the last two columns.

On these instances most versions of our approach solve at least as many problems as the state-of-the-art tools and in significantly less time. Due to the background theory of these instances it is enough to find a propositional model, i.e., solve a MaxSAT problem, since every propositional solution also happens to be T-satisfiable. This is reflected in the last two columns, where the two instantiations of our framework show different behaviour. For **maxhs-msat**, which combines maxHS with the SMT solver, the number of iterations is always one (in all 5319 satisfiable instances). In this case maxHS finds an optimal Boolean model (through several iterations of its *internal* SAT solver), which the SMT solver then verifies to be theory consistent in one call. In case of **cplex-msat** there is no additional SAT solver between the SMT and the optimization components. Therefore it has to learn all the necessary transitivity properties of the equalities in form of cores from the SMT solver. Thus the number of unsatisfiable cores is higher for **cplex-msat**, which can significantly increase solving time depending on the type of hitting sets used.

Table 3.6: Results of considered solvers and configurations on rand-wwtp family with 10%-100% random unit soft clauses. Each %-group consists of 312 problems.

Solver	10%	25%	50%	100%	Total	SMT%	OPT%
cplex-msat	289	271	**203**	4	**767**	60.85%	38.46%
cplex-msat-djnt	286	247	114	2	649	97.35%	1.96%
cplex-msat-min	282	244	142	**16**	684	91.46%	7.68%
cplex-msat-lp	287	262	184	13	746	83.4%	15.27%
maxhs-msat	288	270	179	0	737	42.28%	57.31%
maxhs-msat-djnt	289	249	112	1	651	93.91%	5.69%
maxhs-msat-min	281	242	132	15	670	87.99%	11.59%
maxhs-msat-ll	266	166	16	0	448	7.69%	84.93%
maxhs-msat-ll-djnt	266	161	9	0	436	11.30%	77.59%
maxhs-msat-ll-min	263	166	27	0	456	11.36%	68.11%
optimathsat-maxres	291	258	123	0	672	–	–
optimathsat-omt	240	130	0	0	370	–	–
z3-maxres	280	224	103	0	607	–	–
z3-wmax	**304**	**288**	4	0	596	–	–

These benchmarks in essence allow us to compare the effectiveness of the optimization components independently of the SMT component. This benefits our hitting set based methods, while other solvers rely on alternative approaches. Another important difference is that our prototypes solve lexicographic problems as single objective functions in one run by aggregating the cost functions [175].

The last table (Table 3.6) presents results for the randomized rand-wwtp benchmarks on which **cplex-msat** performs better than **maxhs-msat**. Using non-minimum hitting sets measurably reduces the performance of both implementations on these instances. From the last two columns we can deduce that the best performing methods are those where more time was spent within the optimization component. Although lemma lifting does result in significant more time spent in maxHS calls, some part of it is spent in the SAT solver, and not in actual optimization. This might explain its bad performance.

To summarize, the experiments support the need for a generic framework for MaxSMT. More concretely we make the following three observations. First, there is no overall best configuration. Performance depends on the distribution of the workload among the involved components, since in general the difficulty of the optimization and SMT problems differ. For instance, improved MaxSAT performance does not necessarily translate into improved MaxSMT performance, simply because of different relative costs between SMT calls and SAT calls. Accordingly, non-minimum hitting sets (like disjoint cores or LP relaxation) usually reduce the workload of the optimizer but put more stress on the SMT solver.

Second, the number of extracted unsatisfiable cores or calculated optimal hitting sets is not always an expedient metric to measure the performance of MaxSMT. Finally, most of the time, lemma lifting does not improve but actually seems to reduce performance of a modular MaxSMT solver, particularly with an implicit hitting set based approach. All of our experimental results as well

as the evaluated benchmarks are available at `http://fmv.jku.at/maxsmt/`.

3.7 Conclusion

We have proposed an abstract framework to gain a unifying view of how optimization, propositional reasoning, and theory reasoning can be combined in IHS based MaxSMT solving. Our framework is very flexible supporting a rich space of possible implementation architectures all of which are provably sound. Our empirical results show that different architectures yield quite different performance on different problems sets. This implies that there is considerable potential in more fully exploiting the flexibility of our framework to obtain improved and more robust performance in MaxSMT solvers.

Acknowledgments

This research has been supported by the Austrian Science Fund (FWF) under projects W1255-N23 and S11408-N23.

Chapter 4

Incremental Inprocessing in SAT Solving

Published

In Proceedings of the 22nd International Conference on Theory and Applications of Satisfiability Testing (SAT 2019), pages 136-154, Lisbon, Portugal. See [94].

Authors

Katalin Fazekas, Armin Biere and Christoph Scholl.

Abstract

Incremental SAT is about solving a sequence of related SAT problems efficiently. It makes use of already learned information to avoid repeating redundant work. Also preprocessing and inprocessing are considered to be crucial. Our calculus uses the most general redundancy property and extends existing inprocessing rules to incremental SAT solving. It allows to automatically reverse earlier simplification steps, which are inconsistent with literals in new incrementally added clauses. Our approach to incremental SAT solving not only simplifies the use of inprocessing but also substantially improves solving time.

4.1 Introduction

Solving a sequence of related SAT problems incrementally [11,19,88,131] is crucial for the efficiency of SAT based model checking [40,52,89,154], and important in many domains [109,178,183,215]. Utilizing the effort already spent on a SAT problem by keeping learned information (such as variable scores and learned clauses) can significantly speed-up solving similar problems. Equally important are formula simplification techniques such as variable elimination, subsumption, self-subsuming resolution, and equivalence reasoning [16,86,118,137].

These simplifications are not only applied before the problem solving starts (*preprocessing*), but also periodically during the actual search (*inprocessing*) [138]. In this paper we focus on how to efficiently combine simplification techniques with incremental SAT solving.

Consider the SAT problem $F^0 = (a \vee b) \wedge (\neg a \vee \neg b)$. Both clauses are redundant and can be eliminated by for instance variable or blocked clause elimination [86,137]. The resulting empty set of clauses is of course satisfiable and the SAT solver could for example simply just assign *false* to both variable as a solution. That is of course not a satisfying assignment of F^0, but can be transformed into one by solution reconstruction [136,138], taking eliminated clauses into account. As we will see later, this would set the truth value of either a or b to true.

Now consider the SAT problem $F^1 = (a \vee b) \wedge (\neg a \vee \neg b) \wedge (\neg a) \wedge (\neg b)$ which is actually an extension of F^0 with the clauses $(\neg a)$ and $(\neg b)$. Simply adding them to our simplified F^0 (i.e. to the empty set of clauses) would result in a formula that again is satisfied by assigning false to each variable. However, using solution reconstruction on that assignment leads to the same solution as before, one that satisfies $(a \vee b)$, and thus would actually falsify $(\neg a)$ or $(\neg b)$. The solver would incorrectly report that F^1 is satisfiable, and even return an invalid solution. Thus naively using inprocessing in an incremental setting is not sound.

Obviously one can just give up on incrementality and simply solve F^1 from scratch but with pre- and inprocessing. Another trivial approach is to use learned information from solving F^0, but then disable inprocessing. A compromise is to disallow inprocessing partially by *freezing* [89] those variables that are not allowed to be involved in simplifications ("Don't Touch" variables in [154]). This is rather error-prone and cumbersome for the user, and even often impossible [185].

Our approach benefits from most inprocessing techniques, without freezing any variables. It identifies potential problems between an eliminated clause, such as $(a \vee b)$ in the example, and new clauses, such as $(\neg a)$ and $(\neg b)$. In such a case it moves back the eliminated clause to the formula before adding the new clauses. This greatly simplifies the way how incremental SAT solvers can be used.

The specialized approach in [185] focuses on three preprocessing techniques (variable elimination, subsumption and self-subsumption of [86]). It applies a

preprocessing phase before each incremental SAT call. Instead of that, we adapt and extend the framework of [138] and present a generic calculus which allows to combine a much broader set of pre- and inprocessing techniques with incremental SAT solving. Actually, we use the *most general redundancy property* [124, 125] that covers not only all techniques in [185], but also provides optimized procedures for equivalence literal reasoning [16] and even blocked clause elimination [137]. However, we do not yet support techniques that remove models, such as blocked clause addition [10, 138, 153, 173] (neither does [185]).

Our approach is also more precise than [185] since it allows to distinguish simplification steps applied on different phases of variables, i.e. we provide a literal- and not just variable-based approach. On the practical side, beyond enabling a wider range of pre- and inprocessing techniques, we present a simple algorithm, which yields an efficient implementation as confirmed by our experiments. Using dedicated techniques for inprocessing under assumptions, as [186] extends [185] based on [184], is orthogonal to the approach presented in this paper.

After preliminaries we present our new rules for incremental SAT solving in Sect. 4.3 which are proven correct in Sect. 4.4. We discuss implementation details in Sect. 4.5 followed by experimental results in Sect. 4.6 before we conclude in Sect. 4.7.

4.2 Preliminaries

Satisfiability A *literal* is either a Boolean variable (v), or its negation ($\neg v$). A *clause* is a disjunction of literals, and a *formula* in conjunctive normal form (CNF) is a conjunction of clauses. If convenient, we consider a clause as a set of literals and a formula as a set of clauses. A (partial) *truth assignment* τ is a consistent set of literals assigning truth values to variables as follows. In case $v \in \tau$, then v is assigned *true* by τ (denoted as $\tau(v) = \top$), while if $\neg v \in \tau$, then v is assigned *false* ($\tau(v) = \bot$). A truth assignment *satisfies* a literal ℓ (denoted as $\tau(\ell) = \top$) if $\ell \in \tau$ and it *falsifies* it (denoted as $\tau(\ell) = \bot$) if $\neg\ell \in \tau$, where $\neg\ell = \neg v$ if $\ell = v$ and $\neg\ell = v$ if $\ell = \neg v$. Neither satisfied nor falsified literals are *undefined*. A clause is a *tautology* if it contains both a literal and the negation of it. The application of a truth assignment τ to an arbitrary formula F, denoted as $\tau(F)$ or $F_{|\tau}$, is defined as usual. When it is convenient, we will use sets of literals directly as truth assignments. We further use $\tau_1 \circ \tau_2$ to denote the composition of truth assignments τ_1 and τ_2 in the natural way, i.e., $(\tau_1 \circ \tau_2)(F) = \tau_1(\tau_2(F))$.

The *satisfiability problem* (SAT) for a CNF asks whether there is a truth assignment such that all clauses contain at least one satisfied literal. A truth assignment satisfying a formula is also called a *model*. Formulas F_1, F_2 are *logically equivalent*, denoted as $F_1 \equiv F_2$, if they are satisfied by exactly the same truth assignments, while they are *satisfiability equivalent*, denoted as $F_1 \equiv_{sat} F_2$,

if both of them are satisfiable or both of them are unsatisfiable.

Incremental SAT problems An incremental SAT problem $\mathcal{F}$ is a sequence of clause sets $\langle \Delta_0, \ldots, \Delta_n \rangle$. In phase $i = 0, \ldots, n$ the task is to determine the satisfiability of $F^i = \wedge_{s=0\ldots i} \Delta_s$, the conjunction of all added clauses up to this point. If F^i is unsatisfiable, then F^j for all $j > i$ is unsatisfiable as well, as each iteration just augments the set of clauses. The focus of this paper is on the case where F^i is satisfiable. We rely on the common approach to always choose the given assumptions, literals that are assumed to be true in a phase, as first decisions during search and thus w.l.o.g. do not need to consider assumptions in this paper explicitly. See Minisat [88] for implementation details or [46, 93] for abstract solvers following that approach. However, the variables of assumptions are not allowed to be eliminated or occur in witnesses (e.g., as blocking literal in blocked clauses [137]), i.e., they have to be considered *frozen* [89,154] internally.

Example 4.2.1. *Consider the incremental SAT problem* $\mathcal{F} = \langle \{(a \vee b)\}, \{a, b\} \rangle$. *It consists of two SAT queries:* $F^0 = (a \vee b)$ *and* $F^1 = F^0 \wedge a \wedge b = (a \vee b) \wedge a \wedge b$.

Redundancy in SAT Inprocessing in SAT solving relies on the concept of adding and removing *redundant* clauses. To simplify matters, in this paper we use the most general redundancy notion [124, 125]. It covers most techniques used in current SAT solvers including resolution asymmetric tautology (RAT), which was used in the original work on inprocessing [138]. As [124, 125] points out, any clause redundancy can produce a "witness", e.g. a blocking literal in case of a blocked clause, which allows polynomial solution reconstruction. The following two essential definitions are adapted from [124, 125]:

Definition 4.2.1 (Witness Labelled Clause)**.** *A set of literals* ω *and a clause* C *such that* $\omega \cap C \neq \emptyset$ *is called a witness labelled clause and written as* $(\omega : C)$.

A witness is a set of literals and can be interpreted as a partial truth assignment. With this interpretation, the truth assignment α which falsifies a given clause C but is undefined otherwise is also written as $\alpha = \neg C$.

Definition 4.2.2 (Clause Redundancy)**.** *A witness labelled clause* $(\omega : C)$ *is redundant with respect to a formula* F *if* $\omega(C) = \top$ *and* $F|_\alpha \models F|_\omega$ *for* $\alpha = \neg C$. *This is also denoted as* $F \wedge C \equiv^{\omega}_{sat} F$.

As has been shown in [124,125], this is the most general notion of redundancy and allows to simulate all other types of clause redundancy. The corresponding proof (of Thm. 1 in [124, 125]) allows to "fix" an assignment using the witness. We formalize that part of the proof and extend it to *partial* truth assignments, which allows to use partial truth assignments in the witness reconstruction process satisfying only the simplified formula and is further useful to produce a partial satisfying assignment after reconstruction (used for instance in [20]).

$$\frac{\varphi\,[\,\rho\,]\,\sigma}{\varphi\,[\,\rho\wedge C\,]\,\sigma}\;\natural \qquad \frac{\varphi\,[\,\rho\wedge C\,]\,\sigma}{\varphi\,[\,\rho\,]\,\sigma} \qquad \frac{\varphi\,[\,\rho\wedge C\,]\,\sigma}{\varphi\wedge C\,[\,\rho\,]\,\sigma} \qquad \frac{\varphi\wedge C\,[\,\rho\,]\,\sigma}{\varphi\,[\,\rho\wedge C\,]\,\sigma\cdot(l:C)}\;\flat$$

LEARN FORGET STRENGTHEN WEAKEN

where $\natural$ is "C has RAT w.r.t. $\varphi\wedge\rho$" and $\flat$ is "C has RAT on l w.r.t. φ".

Figure 4.1: Instantiated (with RAT) inprocessing rules as introduced in [138]

Proposition 4.2.1. *Assume $F\wedge C\equiv^{\omega}_{sat} F$ as above. Let τ be a (partial) truth assignment with $\tau(F)=\top$ and $\tau(C)\neq\top$. Then $\gamma(F\wedge C)=\top$ with $\gamma=\tau\circ\omega$.*

Proof. Clearly $\gamma(C)=\omega(C)=\top$. We need to show $\gamma(D)=\top$ for all $D\in F$. Observe $\alpha\circ\tau=\tau\circ\alpha$ with $\alpha=\neg C$ since $\tau(C)\neq\top$ and α and τ are consistent. Thus, $\top=\tau(F)=(\alpha\circ\tau)(F)=(\tau\circ\alpha)(F)=(\tau\circ\alpha)(F\wedge\neg C)=\tau(F|_\alpha)$ since $\alpha(\neg C)=\top$. Using $F|_\alpha\models F|_\omega$ and because $F|_\omega$ remains satisfied for all extensions of τ, we get $\top=(\beta\circ\tau)(F|_\omega)=(\beta\circ\tau\circ\omega)(F)=(\beta\circ\gamma)(F)$, where $\beta=\neg D$ is the truth assignment falsifying the clause $D\in F$, which in particular gives $(\beta\circ\gamma)(D)=\top$. Since $\beta(D)=\bot$ we obtain $\top=(\beta\circ\gamma)(D)=\gamma(D)$. □

Inprocessing Our goal is to adjust and extend the abstract framework of [138] such that incremental SAT solving with inprocessing can be handled. The derivation performed by an inprocessing SAT solver is modelled as a sequence of abstract states. Each state consist of three components: a set of irredundant clauses φ that the solver aims to satisfy, a set of redundant clauses ρ that can be removed without changing the satisfiability of the formula under consideration and an ordered sequence of witness labelled clauses σ (that are actually just literal-clause pairs in [138]), to keep track of eliminated clauses.

To make the paper more self contained Fig. 4.1 lists the original rules of [138], together with the proposed RAT instantiation of side conditions $\natural$ and $\flat$. Rule STRENGTHEN strengthens the irredundant set of clauses, by moving a clause from the redundant set into it, while rule FORGET allows to eliminate a redundant clause from ρ. Rule LEARN introduces a new clause C into the redundant set of clauses in case C has RAT w.r.t. $\varphi\wedge\rho$. Rule WEAKEN simplifies the irredundant set by eliminating a clause C from it if C has RAT on a literal l of C w.r.t. φ. The eliminated clause is moved to the end of the literal-clause pair sequence σ.

Model Reconstruction One challenge of using inprocessing is to guarantee that a satisfying assignment of the final formula can be transformed to a satisfying assignment of the original, non-processed formula. A sequence of witness labelled clauses σ is used as part of the abstract state to keep track of clauses eliminated by WEAKEN during inprocessing. The process of solution reconstruction described through pseudo code in [138] can be formalized as follows:

Definition 4.2.3 (Reconstruction Function)**.** *Given a truth assignment τ and a sequence of witness labelled clauses σ, the reconstruction function is defined as*

$$\mathcal{R}(\tau, \varepsilon) = \tau, \qquad \mathcal{R}(\tau, \sigma \cdot (\omega : D)) = \begin{cases} \mathcal{R}(\tau, \sigma) & \text{if } \tau(D) = \top \\ \mathcal{R}((\tau \circ \omega), \sigma) & \text{otherwise.} \end{cases}$$

The reconstruction function takes a (partial) truth assignment τ and a sequence of witness labelled clauses σ as inputs. It traverses σ in reverse order and sets truth values of those literals in τ to true that are witnesses of not yet satisfied clauses in σ. We are now ready to formalize the central concept of this paper:

Definition 4.2.4 (Reconstruction Property)**.** *A sequence of witness labelled clauses σ satisfies the reconstruction property w.r.t. a formula F iff for all truth assignments τ satisfying F, the result of the reconstruction function $\mathcal{R}$ on τ and σ is a satisfying assignment for $F \wedge \sigma$. An abstract state $\varphi\,[\,\rho\,]\,\sigma$ satisfies the reconstruction property iff σ satisfies the reconstruction property w.r.t. φ.*

For the expression $F \wedge \sigma$ in this definition we interpret σ as a set of its clauses.

4.3 Inprocessing Rules for Incremental Solving

Our first goal is to determine how information, such as learned clauses, can be transferred from one incremental solving phase to the next, utilizing that the sub-problem F^{i+1} is an extension of the previously solved sub-problem F^i. Thus, instead of solving F^{i+1} from scratch, previously learned facts are reused to avoid repeated work. This is sound if the incremental approach gives the same answer (satisfiable or unsatisfiable) as solving from scratch.

More formally, it is crucial that $\varphi^i_{k_i} \wedge \Delta_{i+1} \equiv_{sat} F^{i+1}$ holds, where $\varphi^i_{k_i}$ is the set of irredundant clauses at the end of the evaluation of F^i. We also need to make sure that $F^{i+1} \models \rho^i_{k_i}$, i.e. the redundant clause set at the end of the evaluation of F^i can be reused. Furthermore, we need to guarantee that a model for F^{i+1} can be resconstructed from any satisfying assignment of $\varphi^{i+1}_{k_{i+1}}$.

To establish notation and to emphasize what we would like to improve in this paper, we briefly describe how inprocessing in a non-incremental solver (as in e.g. [36] with cloning) would look like using only the original inprocessing rules of [138] (shown in Fig. 4.1). Each phase $i = 0, \ldots, n$ of solving an incremental problem $\mathcal{F}$ consists of a derivation of a formula $\varphi^i_{k_i} \wedge \rho^i_{k_i}$ as a sequence of states $\langle \varphi^i_0\,[\,\rho^i_0\,]\,\sigma^i_0, \ldots, \varphi^i_{k_i}\,[\,\rho^i_{k_i}\,]\,\sigma^i_{k_i} \rangle$, where (for all $j = 1, \ldots, k_i$)

(a) $\varphi^0_0 = F^0,\ \rho^0_0 = \emptyset,\ \sigma^0_0 = \varepsilon$

(b) $\varphi^i_{j-1}\,[\,\rho^i_{j-1}\,]\,\sigma^i_{j-1}$ results in $\varphi^i_j\,[\,\rho^i_j\,]\,\sigma^i_j$ as application of a rule in Fig. 4.1

(c) $\varphi^{i+1}_0 = F^{i+1},\ \rho^{i+1}_0 = \emptyset,\ \sigma^{i+1}_0 = \varepsilon.$

The *initial state* defined in (a) starts the derivation with F^0 as irredundant set of clauses, with an empty σ and without any redundant clause. Then following (b) the solver applies the rules of Fig. 4.1 until it reaches a state $\varphi^i_{k_i}\ [\,\rho^i_{k_i}\,]\ \sigma^i_{k_i}$ in which satisfiability of $\varphi^i_{k_i} \wedge \rho^i_{k_i}$ is determined. The new phase starts by adding a set of clauses to the problem, as described by (c). Such a derivation only relies on the original rules of [138], so each phase has to restart with completely empty ρ and σ and no information learned from solving F^i can be reused to solve F^{i+1}.

To capture inprocessing in an incremental solver we have to extend and modify the calculus of [138] (in Fig. 4.1). The initial state in (a) and the components of abstract states remain the same as in [138] (see Sect. 4.2), except that σ is more general. In our new calculus it consists of witness labelled clauses instead of literal-clause pairs, which allows to capture any redundancy property (not just RAT). We will refer on that component of a state as *reconstruction stack*.

Next sections describe the derivations of $\varphi^i_{j+1}\ [\,\rho^i_{j+1}\,]\ \sigma^i_{j+1}$ from $\varphi^i_j\ [\,\rho^i_j\,]\ \sigma^i_j$ for each $0 \leq j < k_i$ in each phase $i = 0, \ldots, n$ and show a sound way to start a new phase $i+1$ from state $\varphi^i_{k_i}\ [\,\rho^i_{k_i}\,]\ \sigma^i_{k_i}$ when adding Δ_{i+1} to $\varphi^i_{k_i}$.

4.3.1 Constrained Learning

The side condition $\sharp$ of rule Learn in Fig. 4.1 allows to learn clauses that remove models of the current formula. However, as the following example demonstrates, this is not correct in the context of incremental solving.

Example 4.3.1. *Consider the incremental SAT problem* $\mathcal{F} = \langle\{(a \vee b)\}, \{a, b\}\rangle$. *First in phase* $i = 0$ *the evaluation of* F^0 *starts from the initial state* $(a \vee b)\ [\,\emptyset\,]\ \varepsilon$. *Now the clause* $(\neg a \vee \neg b)$ *can be learned since it has the RAT property w.r.t.* $(a \vee b)$ *(this is* $\sharp$ *in Fig. 4.1). Then, rule* Strengthen *can be applied on* $(\neg a \vee \neg b)$ *which yields state* $(a \vee b) \wedge (\neg a \vee \neg b)\ [\,\emptyset\,]\ \varepsilon$, *with a satisfiable set of irredundant clauses. In the next phase* $i = 1$ *we add* Δ_1 *and target to solve the formula* $F^1 = (a \vee b) \wedge a \wedge b$, *which still is satisfiable. However, conjoining* Δ_1 *to the irredundant clause set of the last state of the previous phase leads to the state* $(a \vee b) \wedge (\neg a \vee \neg b) \wedge a \wedge b\ [\,\emptyset\,]\ \varepsilon$ *with an unsatisfiable irredundant clause set.*

Thus in our calculus the precondition of learning (Learn$^-$) is $\varphi \wedge \rho \models C$, i.e. we allow to learn only *implied* clauses. Compared to [138] our new rule Learn$^-$ is weaker due to this stronger side condition. It still covers most learning techniques in current SAT solvers, except forms of extended resolution such as blocked clause addition [10, 138, 153, 173]. Learned clauses can be forgotten (Forget) or moved to the irredundant formula (Strengthen) as in [138].

4.3.2 Stronger Weakenings

We decompose the original weakening rule (Weaken in Fig. 4.1) of [138] into two rules: Weaken$^+$, as the name suggests, weakens the current formula by

$$\frac{\varphi \wedge C \,[\,\rho\,]\, \sigma}{\varphi \,[\,\rho\,]\, \sigma \cdot (\omega : C)}\ \text{þ} \qquad \text{WEAKEN}^+$$

$$\frac{\varphi \wedge C \,[\,\rho\,]\, \sigma}{\varphi \,[\,\rho\,]\, \sigma}\ \text{ø} \qquad \text{DROP}$$

where þ is $\varphi \wedge C \equiv^{\omega}_{sat} \varphi$ and ø is $\varphi \models C$

Figure 4.2: New weakening and dropping rules

eliminating a clause C from the irredundant set while pushing it to the reconstruction stack. The DROP rule allows to weaken the current formula by eliminating an implied clause from the irredundant set. Removal of implied clauses from φ does not introduce (nor remove) models and so it is not necessary to save these clauses on the reconstruction stack. In our implementation the DROP rule is also used for more advanced equivalence-literal reasoning techniques [9,16,55]. Further, in current implementations weakening is always immediately followed by a forget step (simulating WEAKEN$^+$).

4.3.3 Incremental Clause Addition

The main feature of incremental SAT solving is the possibility to extend the previously solved formula with a set of new clauses. In non-incremental SAT solving, clauses determined to be redundant, always remain redundant. In incremental SAT solving arbitrary clauses can be added and thus previous simplifications might need to be reconsidered and potentially reversed.

Example 4.3.2. *Consider the incremental SAT problem* $\mathcal{F} = \langle \{F^0\}, \{(\neg a \vee b)\}\rangle$*, where* $F^0 = (a \vee b) \wedge (\neg a \vee \neg b) \wedge (a \vee \neg b)$ *and* $F^1 = F^0 \wedge (\neg a \vee b)$*. Phase* $i = 0$ *starts from the state* $(a \vee b) \wedge (\neg a \vee \neg b) \wedge (a \vee \neg b)\ [\,\emptyset\,]\ \varepsilon$*. Resolving the first clause on* a *always produces tautological resolvents (i.e. it is blocked [137]). Thus* WEAKEN$^+$ *can be applied with witness* a*. Afterwards no other irredundant clause contains literal* b *and so both remaining irredundant clauses are blocked on* $\neg b$*. Thus they can be eliminated by* WEAKEN$^+$ *too, which results in state* $\emptyset\ [\,\emptyset\,]\ (a : (a \vee b)) \cdot (\neg b : (\neg a \vee \neg b)) \cdot (\neg b : (a \vee \neg b))$*, without irredundant clauses left, and the solver concludes* F^0 *to be satisfiable. Adding the new clause* $(\neg a \vee b)$ *to incrementally solve* F^1 *yields a state with a satisfiable set of irredundant clauses. But* F^1 *is actually unsatisfiable, so just adding* $(\neg a \vee b)$ *is not sound.*

There are different ways to avoid unsoundness. An obvious way is to simply disallow simplifications over variables (or actually literals in our calculus) that might occur in later phases. In essence, this is the solution implemented through freezing in current SAT solvers [89], which ensures that the reconstruction stack does not contain frozen variables as witnesses. These frozen variables are then the only variables of the current formula that are allowed to reoccur in new clauses. We capture this property as follows.

$$\frac{\varphi\ [\,\rho\,]\ \sigma}{\varphi \wedge \Delta\ [\,\rho\,]\ \sigma}\ \boxed{\mathcal{I}}$$

AddClauses

where $\boxed{\mathcal{I}}$ is that each clause of Δ is clean w.r.t. σ

Figure 4.3: New rule to capture clause set augmentation

Definition 4.3.1 (Clean Clause)**.** *A clause C is clean w.r.t. a sequence of witness labelled clauses σ iff for all $(\omega : D) \in \sigma$ we have that $\neg C \cap \omega = \emptyset$.*

Example 4.3.3. *The clause $(a \vee b)$ is not clean w.r.t. $(\neg b : (\neg a \vee \neg b)) \cdot (\neg b : (a \vee \neg b))$ because $\neg(a \vee b) \cap (\neg b) \neq \emptyset$. On the other hand, $(\neg a \vee \neg b)$ is clean w.r.t. the witness labelled clause sequence $(\neg b : (a \vee \neg b))$ since $\neg(\neg a \vee \neg b) \cap (\neg b) = \emptyset$.*

With this definition the freezing approach guarantees that every added clause is clean w.r.t. the reconstruction stack. Building on that observation, we can now introduce clause addition (AddClauses in Fig. 4.3), where the side condition requires that each new clause in Δ is clean w.r.t. the reconstruction stack σ. If the added clauses are clean w.r.t. the reconstruction stack, then every assignment satisfying them will remain satisfying after applying the reconstruction function:

Lemma 4.3.1. *If a clause C is clean w.r.t. a sequence of witness labelled clauses σ, then for all truth assignments τ with $\tau(C) = \top$ we have that $\mathcal{R}(\tau, \sigma)(C) = \top$.*

Proof. By induction on the length of σ. The base case $\sigma = \varepsilon$ is trivial. Now consider $\sigma \cdot (\omega : D)$ and $\tau' = \tau$ if $\tau(D) = \top$, $\tau' = \tau \circ \omega$ otherwise. If $\tau(D) = \top$, then $\tau'(C) = \tau(C) = \top$. For $\tau(D) \neq \top$ there is $\ell \in C$ with $\tau(\ell) = \top$. As C is clean w.r.t. $(\omega : D)$, i.e., $\neg C \cap \omega = \emptyset$, we have $\neg\ell \notin \omega$ and so $\tau'(\ell) = (\tau \circ \omega)(\ell) = \tau(\ell) = \top$. This also holds if $\ell \in \omega$, since then $\omega(\ell) = \top$. Now it follows by induction applied to τ' and σ: $\top = \mathcal{R}(\tau', \sigma)(C) = \mathcal{R}(\tau, \sigma \cdot (\omega : D))(C)$. □

Thus, as long as all our clause elimination steps are based on witnesses that never occur in new clauses, we can add clauses without any problem in new incremental calls. However, this approach requires to know in advance in every phase i every literal of every Δ_j with $j > i$. Beyond that, it allows less clauses to be eliminated. Fortunately we can do better.

Instead of prohibiting simplifications, we allow arbitrary inprocessing as in a non-incremental SAT solver, but later reverse simplifications inconsistent with new clauses. It would be easy to just reverse all simplifications by reintroducing all eliminated clauses, but this is costly (as our experiments show). Therefore, it would be desirable to reverse a minimal subset of simplifications, but such a minimal set is in general difficult to identify.

As compromise we try to cheaply identify a sufficient subset of problematic simplifications as follows. If a new clause is not clean w.r.t. the reconstruction stack, we reverse those simplifications which have a negated literal of the

$$\frac{\varphi\,[\,\rho\,]\,\sigma\cdot(\omega:C)\cdot\sigma'}{\varphi\wedge C\,[\,\rho\,]\,\sigma\cdot\sigma'}\;\boxed{\partial}$$

RESTORE

where $\boxed{\partial}$ is C is clean w.r.t. σ'

Figure 4.4: New rule to reverse a weakening step

new clause in the witness. Reversing all these problematic steps yields a clean reconstruction stack for all new clauses that in turn allows to apply rule ADDCLAUSES.

4.3.4 Reversing Weakening

The side condition of rule ADDCLAUSES identifies which simplifications need to be reversed in order to add a set of new clauses to the formula. What is missing is a rule to actually reverse these steps. The challenge with reversing clause eliminations is that many simplification steps are dependent on each other, e.g., in F^0 of Ex. 4.3.2 the last two clauses became blocked only after the first simplification step. Therefore one can not just arbitrarily reverse simplifications:

Example 4.3.4. *Consider again the inprocessing of F^0 in Example 4.3.2, with the final state $\emptyset\,[\,\emptyset\,]\,(a:(a\vee b))\cdot(\neg b:(\neg a\vee\neg b))\cdot(\neg b:(a\vee\neg b))$. Assume we reverse the first simplification step, i.e., we move $(a\vee b)$ from the reconstruction stack to the irredundant clauses. The truth assignment $\tau=\{\neg a,b\}$ would satisfy in the resulting state $(a\vee b)\,[\,\emptyset\,]\,(\neg b:(\neg a\vee\neg b))\cdot(\neg b:(a\vee\neg b))$ the irredundant clauses. The reconstruction function on that assignment and the current stack would be $\mathcal{R}(\tau,(\neg b:(\neg a\vee\neg b))\cdot(\neg b:(a\vee\neg b)))$. Since $\tau(a\vee\neg b)\neq\top$, it first updates τ with the witness of that clause and becomes $\tau'=(\tau\circ\{\neg b\})=\{\neg a,\neg b\}$. Then, τ' satisfies the next clause of the stack and so $\mathcal{R}(\tau',(\neg b:(\neg a\vee\neg b)))=\mathcal{R}(\tau',\varepsilon)=\tau'$. However, $\tau'(a\vee b)=\bot$. Thus, reversing only the first simplification step led to a state where we failed to reconstruct a solution for F^0.*

Our main contribution is the rule RESTORE in Fig. 4.4 which provides a sound way to reintroduce *selected* clauses from the stack back to the formula using the concept of clean clauses of Def. 4.3.1 as precondition.

Example 4.3.5. *Consider again formula F^0 of Example 4.3.2. Example 4.3.3 shows that the first clause of the stack is not clean w.r.t. its suffix ($(a\vee b)$ w.r.t. $(\neg b:(\neg a\vee\neg b))\cdot(\neg b:(a\vee\neg b))$), but the second and third clauses are both clean ($(\neg a\vee\neg b)$ w.r.t. $(\neg b:(a\vee\neg b))$ and $(a\vee\neg b)$ w.r.t. ε). Restoring the second clause leads to the state $(\neg a\vee\neg b)\,[\,\emptyset\,]\,(a:(a\vee b))\cdot(\neg b:(a\vee\neg b))$. A satisfying assignment of $(\neg a\vee\neg b)$ is $\tau=\{\neg a,\neg b\}$. The reconstruction function on τ and the current stack would be then $\mathcal{R}(\tau,(a:(a\vee b))\cdot(\neg b:(a\vee\neg b)))=\mathcal{R}(\tau,(a:(a\vee b)))$, since $\tau(a\vee\neg b)=\top$. Because $\tau(a\vee b)\neq$*

$$\frac{\varphi\,[\,\rho\,]\,\sigma}{\varphi\,[\,\rho \wedge C\,]\,\sigma}\;\sharp \qquad \text{Learn}^{-}$$

$$\frac{\varphi\,[\,\rho \wedge C\,]\,\sigma}{\varphi\,[\,\rho\,]\,\sigma} \qquad \text{Forget}$$

$$\frac{\varphi\,[\,\rho \wedge C\,]\,\sigma}{\varphi \wedge C\,[\,\rho\,]\,\sigma} \qquad \text{Strengthen}$$

$$\frac{\varphi\,[\,\rho\,]\,\sigma}{\varphi \wedge \Delta\,[\,\rho\,]\,\sigma}\;\mathcal{I} \qquad \text{AddClauses}$$

$$\frac{\varphi \wedge C\,[\,\rho\,]\,\sigma}{\varphi\,[\,\rho\,]\,\sigma \cdot (\omega : C)}\;\flat \qquad \text{Weaken}^{+}$$

$$\frac{\varphi \wedge C\,[\,\rho\,]\,\sigma}{\varphi\,[\,\rho\,]\,\sigma}\;\varnothing \qquad \text{Drop}$$

$$\frac{\varphi\,[\,\rho\,]\,\sigma \cdot (\omega : C) \cdot \sigma'}{\varphi \wedge C\,[\,\rho\,]\,\sigma \cdot \sigma'}\;\partial \qquad \text{Restore}$$

where $\sharp$ is $\varphi \wedge \rho \models C$, $\flat$ is $\varphi \wedge C \equiv^{\omega}_{sat} \varphi$, $\varnothing$ is $\varphi \models C$, ∂ is C is clean w.r.t. σ' and $\mathcal{I}$ is that each clause in Δ is clean w.r.t. σ

Figure 4.5: Incremental inprocessing rules

$\top$, τ *needs to be updated with the witness* a, $\tau' = \tau \circ \{a\} = \{a, \neg b\}$. *Then* $\mathcal{R}(\tau, (a : (a \vee b))) = \mathcal{R}(\tau', \varepsilon) = \tau'$. *The resulting assignment* τ' *satisfies not just the irredundant formula but each clause of the stack as well. Similarly, starting from any other satisfying assignment of* $(\neg a \vee \neg b)$, *the result of the reconstruction function satisfies all clauses.*

4.3.5 Incremental Inprocessing Rules

The final and complete version of our calculus is shown in Fig. 4.5. To keep the notation simple the precise indexing of the states were so far omitted. Following the convention introduced at the beginning of this section, each single-line rule allows to derive a state $\varphi^i_{j+1}\,[\,\rho^i_{j+1}\,]\,\sigma^i_{j+1}$ from a state $\varphi^i_j\,[\,\rho^i_j\,]\,\sigma^i_j$, with $0 \leq i \leq n$ and $0 \leq j < k_i$, while our double-line rule AddClauses transits from a state $\varphi^i_{k_i}\,[\,\rho^i_{k_i}\,]\,\sigma^i_{k_i}$ to state $\varphi^{i+1}_0\,[\,\rho^{i+1}_0\,]\,\sigma^{i+1}_0$.

4.4 Formal Correctness

First we show that learned clauses are still valid in the next phase, and then prove that solutions can be reconstructed in each satisfiable state. In these proofs the set of irredundant clauses are always considered in *combination* together with the clauses on the reconstruction stack, i.e., $\varphi^i_j \wedge \sigma^i_j$. An important finding of our paper is that these combined formulas always imply the redundant clauses.

Proposition 4.4.1. *In any derivation in our calculus starting from the initial state the property* $\varphi^i_j \wedge \sigma^i_j \models \rho^i_j$ *holds for each phase* $i = 0 \ldots n$ *and* j *with* $0 \leq j \leq k_i$.

Proof. In the initial state $\varphi^0_0 \wedge \sigma^0_0 \models \rho^0_0$ trivially holds because ρ^0_0 is empty. Assume that $\varphi^i_j \wedge \sigma^i_j \models \rho^i_j$ holds (for any i and j s.t. $0 \leq i \leq n$ and $0 \leq j < k_i$). We show that any transition maintains the property. In case rule Forget or Strengthen is applied, ρ^i_{j+1} is weaker than ρ^i_j. In case of Forget, $\varphi^i_{j+1} = \varphi^i_j$

and $\sigma^i_{j+1} = \sigma^i_j$, while in case of Strengthen φ^i_{j+1} is even stronger than φ^i_j, and thus $\varphi^i_{j+1} \wedge \sigma^i_{j+1} \models \rho^i_{j+1}$ trivially follows in both cases. Rules Weaken$^+$ and Restore only move a clause between φ^i_j and σ^i_j and so $\varphi^i_j \wedge \sigma^i_j$ remains unchanged. Due to $\boxed{\emptyset}$, in case of Drop, $\varphi^i_j \equiv \varphi^i_{j+1}$, and so it also trivially maintains the property. When Learn$^-$ transits from state j to $j+1$, we get from the inductive assumption that $\varphi^i_j \wedge \sigma^i_j \models \varphi^i_j \wedge \rho^i_j$ and due to $\boxed{\sharp}$, we know that $\varphi^i_j \wedge \rho^i_j \models C$, and so $\varphi^i_j \wedge \sigma^i_j \models \rho^i_j \wedge C = \rho^i_{j+1}$. When starting a new phase (i.e. moving from i to $i+1$ where $0 \leq i < n$ and j is k_i) only new clauses are added to $\varphi^i_{k_i}$ by AddClauses, and so $\varphi^{i+1}_0 \wedge \sigma^{i+1}_0 \models \rho^{i+1}_0$ clearly holds. □

With this proposition we can now prove that the combined formulas remain logically equivalent during a derivation, unless new clauses are added.

Proposition 4.4.2. *In any derivation starting from the initial state, the property $\varphi^i_j \wedge \sigma^i_j \equiv \varphi^i_{j+1} \wedge \sigma^i_{j+1}$ holds for phase $i = 0 \ldots n$ and each j with $0 \leq j < k_i$.*

Proof. Only the rules Strengthen and Drop change the combined formula. However, Strengthen strengthens with an implied clause (due to Prop. 4.4.1), while Drop guarantees logical equivalence due to its side condition. □

From that follows that at any point of a derivation within one phase the combined formula is logically equivalent to the incremental sub-problem:

Corollary 4.4.1. $F^i \equiv \varphi^i_0 \wedge \sigma^i_0 \equiv \varphi^i_1 \wedge \sigma^i_1 \equiv \cdots \equiv \varphi^i_{k_i} \wedge \sigma^i_{k_i}$.

Proof. $F^0 \equiv \varphi^0_0 \wedge \varepsilon \equiv \cdots \equiv \varphi^0_{k_0} \wedge \sigma^0_{k_0}$. By an inductive argument and Prop. 4.4.2: $F^{i+1} = F^i \wedge \Delta_{i+1} \equiv \varphi^i_{k_i} \wedge \sigma^i_{k_i} \wedge \Delta_{i+1} = \varphi^{i+1}_0 \wedge \sigma^{i+1}_0 \equiv \cdots \equiv \varphi^{i+1}_{k_{i+1}} \wedge \sigma^{i+1}_{k_{i+1}}$. □

Moreover, an important practical consequence of Cor. 4.4.1 and Prop. 4.4.1 is that it is sound to keep the learned clauses of the solver when new clauses are added:

Corollary 4.4.2. $F^{i+1} \models \rho^i_{k_i}$.

Before we can prove that we can reconstruct a model for the original incremental problem from a model of the current irredundant clauses using the reconstruction stack we need the following lemma.

Lemma 4.4.3. *For a given truth assignment τ and a sequence of witness labelled clauses $\sigma \cdot \sigma'$ we have $\mathcal{R}(\tau, \sigma \cdot \sigma') = \mathcal{R}(\mathcal{R}(\tau, \sigma'), \sigma)$.*

Proof. By induction over the length of σ'. The base case $\sigma' = \varepsilon$ is trivial. Now consider $\sigma' = \sigma'' \cdot (\omega : C)$ and let $\tau' = \tau$ if $\tau(C) = \top$, $\tau' = \tau \circ \omega$ otherwise. Since $\mathcal{R}(\tau, \sigma \cdot \sigma') = \mathcal{R}(\tau', \sigma \cdot \sigma'')$ and $\mathcal{R}(\tau, \sigma') = \mathcal{R}(\tau', \sigma'')$, $\mathcal{R}(\tau, \sigma \cdot \sigma') = \mathcal{R}(\mathcal{R}(\tau, \sigma'), \sigma)$ follows from the induction hypothesis applied to τ' and $\sigma \cdot \sigma''$. □

Theorem 4.4.4 (Reconstructiveness)**.** *In any derivation starting from the initial state, every state satisfies the reconstruction property of Def. 4.2.4.*

Proof. In the initial state the reconstruction stack is empty, and so for any satisfying assignment τ of F^0, $\mathcal{R}(\tau, \varepsilon)(F^0) = \top$. To simplify notation, we first consider only a single phase i (with $0 \leq i \leq n$), and omit the superscript i. Assume that in a state j (where $0 \leq j < k_i$), the reconstruction property holds. Let τ be a truth assignment with $\tau(\varphi_j) = \top$. Then $\mathcal{R}(\tau, \sigma_j)(\varphi_j \wedge \sigma_j) = \top$ follows by induction. In case LEARN$^-$ or FORGET was applied to state j, we have $\varphi_{j+1} = \varphi_j$ and $\sigma_{j+1} = \sigma_j$, thus the reconstruction property remains true. Rule STRENGTHEN moves a clause C from ρ_j to φ_{j+1} and so $\varphi_{j+1} = \varphi_j \wedge C$ and $\sigma_{j+1} = \sigma_j$. In case $\tau(\varphi_{j+1}) = \top$ we have $\mathcal{R}(\tau, \sigma_{j+1})(\varphi_j \wedge \sigma_{j+1}) = \mathcal{R}(\tau, \sigma_j)(\varphi_j \wedge \sigma_j) = \top$ by induction. Then Prop. 4.4.1 gives $\varphi_j \wedge \sigma_j \models C$, thus $\mathcal{R}(\tau, \sigma_{j+1})(\varphi_j \wedge C \wedge \sigma_{j+1}) = \top$. From the side condition of DROP we know that $\tau(\varphi_{j+1} \wedge C) = \top$ whenever $\tau(\varphi_{j+1}) = \top$, and thus $\mathcal{R}(\tau, \sigma_{j+1})(\varphi_{j+1} \wedge \sigma_{j+1}) = \top$ again by induction. When WEAKEN$^+$ is applied, a redundant clause C is removed from φ_j and pushed to σ_{j+1} (i.e. $\varphi_j = \varphi_{j+1} \wedge C$) witnessed by ω. Assume $\tau(\varphi_{j+1}) = \top$. We apply the induction hypothesis to the truth assignments τ and $(\tau \circ \omega)$ to get:

$$\tau(\varphi_{j+1} \wedge C) = \top \Rightarrow \mathcal{R}(\tau, \sigma_j)(\varphi_{j+1} \wedge C \wedge \sigma_j) = \top \tag{4.1}$$

$$(\tau \circ \omega)(\varphi_{j+1} \wedge C) = \top \Rightarrow \mathcal{R}((\tau \circ \omega), \sigma_j)(\varphi_{j+1} \wedge C \wedge \sigma_j) = \top. \tag{4.2}$$

If $\tau(C) = \top$, then $\mathcal{R}(\tau, \sigma_j \cdot (\omega : C))(\varphi_{j+1} \wedge C \wedge \sigma_j) = \top$ due to (4.1). Furthermore, assuming the side condition of WEAKEN$^+$, we know that $(\omega : C)$ is redundant w.r.t. φ_{j+1}. If $\tau(C) \neq \top$, then $(\tau \circ \omega)(\varphi_{j+1} \wedge C) = \top$ using Prop. 4.2.1. And with (4.2) we also get $\mathcal{R}(\tau, \sigma_j \cdot (\omega : C))(\varphi_{j+1} \wedge C \wedge \sigma_j) = \top$ if $\tau(C) \neq \top$. When we restore a clause C by RESTORE, we know that if $\tau(\varphi_j \wedge C) = \top$ then $\tau(C) = \top$. Further, we know from the side condition of RESTORE that C is clean w.r.t. σ', and so with Lemma 4.3.1, we obtain $\mathcal{R}(\tau, \sigma')(C) = \top$. From that and from Lemma 4.4.3 it follows that $\mathcal{R}(\tau, \sigma \cdot (\omega : C) \cdot \sigma') = \mathcal{R}(\mathcal{R}(\tau, \sigma'), \sigma \cdot (\omega : C)) = \mathcal{R}(\mathcal{R}(\tau, \sigma'), \sigma) = \mathcal{R}(\tau, \sigma \cdot \sigma')$, where $\varphi_j \wedge \sigma \wedge C \wedge \sigma'$ evaluates to true due to the induction hypothesis. When a new phase starts (i.e. $0 \leq i < n$ and $j = k_i$) as Δ_{i+1} is added to $\varphi^i_{k_i}$ by ADDCLAUSES, each new clause is clean w.r.t. $\sigma^i_{k_i}$. Thus, due to Lemma 4.3.1, the reconstruction function does not destroy any satisfying assignment of Δ_{i+1}. □

Theorem 4.4.5 (Correctness)**.** *In any derivation starting from the initial state, for each phase $i = 0 \ldots n$ we have $F^i \equiv_{sat} \varphi^i_j \wedge \rho^i_j$ for all j with $0 \leq j \leq k_i$.*

Proof. From Prop. 4.4.1 and Thm. 4.4.4 it follows, that φ^i_j is unsatisfiable if $\varphi^i_j \wedge \rho^i_j$ is unsatisfiable. In this case also F^i is unsatisfiable using Cor. 4.4.1. Otherwise, if $\varphi^i_j \wedge \rho^i_j$ is satisfiable, then F^i is satisfiable due to Thm. 4.4.4 and again Cor. 4.4.1. □

RestoreAddClauses (new clauses Δ, reconstruction stack σ)

```
1 (ω_1 : C_1) ··· (ω_n : C_n) := σ
2 for i from 1 to n
3     if exists ℓ ∈ ω_i where ¬ℓ occurs in Δ then
4         Δ := Δ ∪ C_i ,   σ := σ \ (ω_i : C_i)
5 return ⟨Δ, σ⟩
```

Figure 4.6: Algorithm RestoreAddClauses to identify and restore all tainted clauses.

To summarize, our calculus fulfills all the desiderata listed at the beginning of Sect. 4.3: (*i*) we can reuse the gained information of previous iterations (including learned clauses), (*ii*) we can continue with incremental solving in a satisfiability preserving way, and (*iii*) the reconstruction property guarantees that we can get a solution to the original problem in case of satisfiability.

4.5 Implementation

Based on our new approach we added incremental inprocessing to the SAT solver CaDiCaL [37]. Rule Weaken$^+$ is defined in our calculus based on the most general redundancy property and so it allows to employ every clause elimination procedure implemented in CaDiCaL including variable elimination [86], vivification [169,197], equivalent-literal substitution [9,55], hyper-binary resolution [16], (self-)subsumption [86] and blocked clause elimination [137]. Combining Drop with Weaken$^+$ allows efficient equivalence literal substitution, since only two binary clauses have to be stored on the stack for each literal in a strongly connected component [9, 55] instead of all clauses with that literal. Similarly, gate-based variable elimination [86] only requires to save gate clauses.

At the heart of our new calculus are the Restore and AddClauses rules. They allow to reverse problematic simplification steps and add new clauses. In practice, SAT solvers are used via an interface (e.g. IPASIR [19] in CaDiCaL) to add new clauses Δ and then asked to solve the extended formula $F \wedge \Delta$. Before solving $F \wedge \Delta$, our approach first performs a sequence of Restore steps in order to make each clause in Δ clean w.r.t. the reconstruction stack σ using the algorithm RestoreAddClauses in Fig. 4.6. Then the new and restored clauses are added to the irredundant clauses and a new incremental solving phase starts.

The algorithm in Fig. 4.6 presents a simple implementation that identifies a sufficient set of clauses to restore in order to make Δ clean. It follows the idea of "taint-checking", commonly used to reason about information-flow (see e.g. [213]). First consider every clause that comes from the user as tainted, because it potentially leads to problems. Then check whether these tainted clauses (actually literals of these clauses) trigger any clause on the stack to be restored. In that case the literals of the restored clause become tainted as well

and recursively might trigger further clauses. However, restored clauses only need to be clean w.r.t. the reconstruction stack after them (see RESTORE in Fig. 4.4), while the clauses in Δ need to be clean w.r.t. the whole reconstruction stack. Therefore, the need for restoring is checked by traversing the stack from bottom to top (left to right). If a clause has to be restored, it can only trigger to restore clauses to its right. Thus, already processed clauses on the left do not have to be reconsidered.

The method takes the new clauses Δ and the current stack σ as input and checks each previous simplification step from left to right (see Line 1-2). Whenever the witness of a simplification has a literal that occurs negated in Δ, the simplification is reversed by restoring the eliminated clause from the stack. The check in Line 3 is actually asking whether there is a clause in Δ (i.e. in the set of new or already restored clauses) that is not clean w.r.t. $(\omega_i : C_i)$. To implement this check efficiently, we mark literals in Δ as tainted and in σ as witness. If the check succeeds, we need to restore the problematic C_i so that at the end we have a clean stack. In Line 4 the restored clause is added to Δ and removed from the stack. At the end of the procedure, Δ contains all the new and restored clauses, which added to the formula together with the new σ achieves the same effect as applying a sequence of RESTORE steps and a final ADDCLAUSES.

4.6 Experiments

We implemented a new bounded model checker called CaMiCaL for AIGER models [41], as used in the hardware model checking competition (HWMCC) [44]. Unrolling is simulated symbolically through substitution [142] in combination with structural hashing [119,152] and local low-level AIG optimizations [59]. As back-end different configurations of our SAT solver CaDiCaL [37] and other state-of-the-art incremental SAT solvers are used. The model checker was run on all the 300 models of the single safety property track of HWMCC'17 [44] up to bound 1000 with a time limit of 3600 seconds (for each model) and memory limit of 8 GB on our cluster with Intel Xeon E5-2620 v4 @ 2.10GHz CPUs.

Results are presented in a similar way as the well-known cactus plots of the SAT Competition, except that we do not measure the overall running time of the model checker, but the time needed for one (incremental) call to the SAT solver. Figure 4.7 shows the distribution of these solving times. For example, if the model checker finished proving unsatisfiability for bound 41 after 110 seconds and then proved unsatisfiability for the consecutive bound 42 at 125 seconds then the time difference of 15 seconds is accounted for bound 42 on this instance. At the end each instance contributes as many solving times as bounds for it are solved.

As expected, worst performance is observed when the SAT solver is used in a completely non-incremental way (cadical-non-incremental), even with pre- and

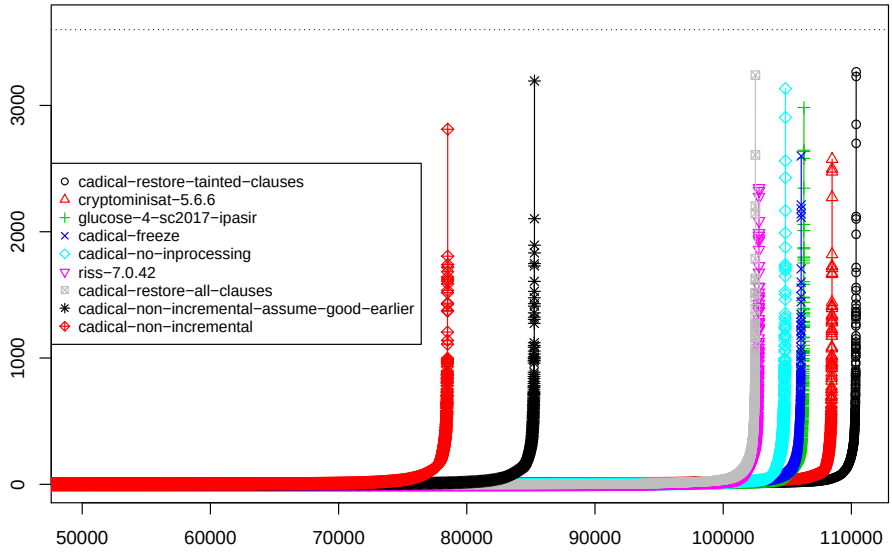


Figure 4.7: Experimental results on all the 300 instances of the single safety property track of HWMCC'17. The x-axis corresponds to all bounds solved over all models sorted by the time needed for the SAT call for each bound, which is on the y-axis. The dotted horizontal line at 3600 second shows the time limit for solving all bounds of each model.

inprocessing enabled. It improves, if the model checker is allowed to assume earlier bounds to be good (cadical-non-incremental-assume-good-earlier). Incremental SAT solving is better as configuration cadical-restore-all-clauses shows, which employs pre- and inprocessing, but at the beginning of incremental calls restores all weakened clauses. However, disabling pre- and inprocessing completely during incremental SAT solving (cadical-no-inprocessing) is even better.

Configuration cadical-freeze can use variables for simplification which are not frozen. This again improves performance and there is no need to restore clauses. In bounded model checking (BMC) only variables encoding the next state are used in future calls and freezing them is sufficient. However, it required substantial programming effort to identify the set of frozen variables. Further, optimizations during CNF encoding, including structural hashing [119, 152] across time frames or local two-level AIG optimizations [59], make it difficult to predict future use of variables. In other cases freezing might not even be possible [185].

Giving up on freezing makes use of our framework and gave the best solving times as configuration cadical-restore-tainted-clauses shows. This not only simplifies the way the solver is used through the API (no need to freeze variables) but also improves solving time. We measured the time spent in RestoreAddClauses to be less than 1% of the overall running time: 0.14% for our best configuration cadical-restore-tainted-clauses and 0.33% for cadical-restore-all-clauses. Our best configuration only restored 17% of the clauses. Restoring all clauses also lead to 3.4 times more eliminated clauses (applications of Weaken$^+$) in total.

Note that one can not get rid of freezing completely, since assumptions (for

the "bad" state property in BMC) have to be frozen internally. Keeping freezing in the API might for instance also be useful for CNF simplification [154].

We also have similar results using freezing (as it is necessary for the solver Lingeling [36]) versus restoring tainted clauses for CaDiCaL as SAT solver back-end of our SMT solver Boolector [191]. We solved more benchmarks and decreased solving time significantly with the consequence that CaDiCaL is likely to replace Lingeling as incremental SAT solver back-end in the future.

We also considered other highly ranked SAT solvers in incremental tracks of the SAT Competition [19, 21, 22]: Glucose 4 [12], CryptoMiniSAT 5.6.6 [22, 223] and Riss 7.0.42 [172]. CryptoMiniSAT performs significantly better than the other two external solvers. It is the only external solver which performs inprocessing during solving, including distillation [116]. Even though CryptoMiniSAT implements the same solution as [185] for incremental bounded variable elimination (BVE), this feature cannot be enabled through the API, and is disabled in our experiments. According to Mate Soos (private communication) scheduling BVE efficiently for incremental SAT solving is difficult for CryptoMiniSAT. We simply schedule BVE in CaDiCaL in the same way as during stand-alone SAT solving, with a persistent schedule across incremental invocations. Note that CaDiCaL only tries to eliminate variables and clauses which are newly added (or restored).

Source code of CaDiCaL and CaMiCaL and experimental data related to Fig. 4.7 can be found at `http://fmv.jku.at/incrinpr` including additional plots.

4.7 Conclusion

This paper presents a calculus that extends the framework of [138] to capture incremental SAT solving. It uses the most general clause redundancy property and is able to simulate most simplifications implemented in state-of-the-art SAT solvers. Our proposed approach is simple, eases the burden of using SAT solvers, can be implemented efficiently, and also reduces solving time substantially. As future work we want to support techniques which remove models, such as blocked clause addition, and techniques for simplifying under assumptions.

Acknowledgments. This research has been supported by the Austrian Science Fund (FWF) under projects W1255-N23 and S11408-N23. We thank Mathias Preiner and Aina Niemetz for their help in experimenting with Boolector and Håkan Hjort for providing feedback on using an incremental version of CaDiCaL.

Chapter 5

Duplex Encoding of Staircase At-Most-One Constraints for the Antibandwidth Problem

Published

In Proceedings of the 17th International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR 2020), to appear. See [97].

Authors

Katalin Fazekas, Markus Sinnl, Armin Biere and Sophie Parragh

Abstract

Decision and optimization problems can be tackled with different techniques, such as Mixed Integer Programming, Constraint Programming or SAT solving. An important ingredient in the success of each of these approaches is the exploitation of common constraint structures with specialized (re-)formulations, encodings or other techniques. In this paper we present a new linear SAT encoding using binary decision diagrams over multiple variable orders as intermediate representation of a special form of constraints denoted as staircase at-most-one constraints. The use of these constraints is motivated by recent work on the antibandwidth problem, where an iterative solution procedure using feasibility-mixed integer programs based on such constraints was most effective. In a computational study we compare the effectiveness of our new encoding against traditional SAT-encodings for staircase at-most-one constraints. Additionally we compare against previous exact solution methods for the antibandwidth problem, such as a constraint programming approach and the one based on feasibility-mixed integer programs.

5.1 Introduction

An important ingredient in the success of computational approaches, such as Mixed Integer Programming (MIP), Constraint Programming (CP) or propositional satisfiability solving (SAT), for solving optimization and decision problems is the exploitation of common constraint structures with specialized encodings, (re-)formulations or other techniques (see e.g. [3, 49, 230]).

In this paper we present a new and specialized SAT encoding of problems where an at-most-one constraint slides over a sequence of Boolean variables. We denote this special case of sliding sequence constraints [29, 33, 57, 229] as *staircase at-most-one constraint* (SCAMO) and illustrate the reason for this name with the following example.

Example 5.1.1. *Given a sequence of variables* $X = \langle x_1\, x_2 \cdots x_{10} \rangle$*, the staircase at-most-one constraint set of width* 4 *is the following formula:*

$$
\begin{array}{llllllllllll}
x_1 & + & x_2 & + & x_3 & + & x_4 & & & & & \leq 1 \wedge \\
 & & x_2 & + & x_3 & + & x_4 & + & x_5 & & & \leq 1 \wedge \\
 & & & & x_3 & + & x_4 & + & x_5 & + & x_6 & \leq 1 \wedge \\
 & & & & & & x_4 & + & x_5 & + & x_6 + x_7 & \leq 1 \wedge \\
 & & & & & & & & x_5 & + & x_6 + x_7 + x_8 & \leq 1 \wedge \\
 & & & & & & & & & & x_6 + x_7 + x_8 + x_9 & \leq 1 \wedge \\
 & & & & & & & & & & x_7 + x_8 + x_9 + x_{10} & \leq 1.
\end{array}
$$

This research is motivated by recent work [220] of the second author on the *antibandwidth problem* (ABP). The ABP is a *graph labeling problem* (see e.g. [99] for more on such problems) where the goal is to maximize the smallest difference between labels of neighbouring nodes. It has various applications, such as scheduling [158], obnoxious facility location [67], radio frequency assignment [115] and map-coloring [100]. It has been studied from a theoretical point of view (see e.g. [28, 84, 180, 205, 232, 235]), and several heuristics and metaheuristics (e.g. [23, 85, 168, 214]) have been designed for it. In [85], aside from a metaheuristic, also a MIP approach was presented to solve the ABP exactly.

In [220] new MIP formulations were presented, and based on one of them, an iterative solution procedure, which repeatedly solved feasbility-MIPs, was designed. For a given number k, these MIPs encode the question whether there exists a solution with antibandwidth greater than k. This iterative procedure actually proved to be the most effective one in the computational study of [220].

Our proposed encoding can be used for more difficult problem structures than the one given in Example 5.1.1. In the ABP, for example, the difference of labels of neighbouring nodes is restricted by combining two SCAMO constraints on two sequences of variables. Aside from the ABP (and other labeling problems), the SCAMO constraints can potentially be used in many further application contexts, such as scheduling problems (see e.g. [51, 174, 228]) or in staff roster-

ing [64, 92] and car sequencing problems [83, 222], when at most one variable is allowed to take a given value in every sequence of variables.

As at-most-one constraints are ubiquitous in applications of SAT they are featured prominently in the literature, see e.g. [70, 130, 147, 173, 189, 203]. They are forming a special case of cardinality constraints [39, 160, 221], which in turn are instances of Pseudo-Boolean constraints [2, 90, 196, 209] and thus 0/1 integer linear programs. Encoding constraints (for an overview see [189]) instead of handling them natively (as in [160]) allows to make full use of the power of SAT solving. For some applications mixed strategies [98] are better though. In practice, size is the most important criteria to evaluate such encodings, while at least in theory also propagation strength is considered. See [1] for a discussion of these trade-offs. In particular, the path based encoding of binary decision diagrams introduced in [1] has the goal to improve propagation. However, as the authors point out, it can not be used for encoding shared constraints, which is the main reason of the efficiency in our encoding. Thus we also provide a new set of benchmarks for which such sharing occurs naturally.

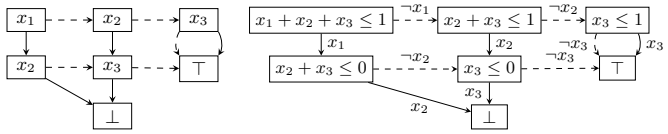
5.2 Preliminaries

A propositional formula in conjunctive normal form (CNF) consists of a set of clauses, where each clause C is a disjunction of literals, which are Boolean (also called 0/1) variables (e.g. x) or their negation ($\neg x$ or $1-x$). A truth assignment T maps truth values (0/1 values) to Boolean variables and can be represented by a set of consistent literals; it satisfies a literal ℓ (i.e. assigns value 1 to ℓ) if $\ell \in T$, and falsifies it (assigns value 0 to ℓ) if $\neg\ell \in T$, where $\neg\ell = \neg x$ if $\ell = x$ and $\neg\ell = x$ if $\ell = \neg x$. The satisfiability problem (SAT) for a formula in CNF asks whether there is a truth assignment such that all clauses contain at least one satisfied literal. A truth assignment satisfying a formula is also called a model.

An *at-most-one* (AMO) constraint is an expression of the form $\sum_{i=1}^{n} x_i \leq 1$, where $x_1, x_2, \ldots, x_n$ are Boolean variables. Similarly, we can formulate *at-most-zero* (AMZ) constraints (as $\sum_{i=1}^{n} x_i \leq 0$), which actually states that each variable must be false (i.e. assigned value 0). Further, an exactly-one (EO) constraint is an expression of the form $\sum_{i=1}^{n} x_i = 1$. Notice that we define and use these constraints over Boolean variables, but they are trivially extensible to literals.

A binary decision diagram (BDD, see e.g. [60,61]) is a rooted, directed, acyclic graph with at most two leafs, labeled with $\bot$ (false or 0) and $\top$ (true or 1). Every non-leaf (also called nonterminal) node of a BDD is labeled with a Boolean variable and has exactly two outgoing edges (called *low* and *high* in [60]). In this paper we use BDDs to represent AMO and AMZ constraints. Figure 5.1a depicts an example BDD of an AMO constraint over variables x_1, x_2 and x_3. Each path from the root of the BDD that ends in the true leaf ($\top$) is a model of $x_1 + x_2 + x_3 \leq 1$. Whenever the low or high child (marked with dashed

resp. solid line in Fig. 5.1) of a node labeled with variable x is taken, it means that x is assigned to be false (true respectively) on that path. Since all our BDDs represent AMO or AMZ constraints, we will depict them rather in an expanded form where each node contains the whole Boolean expression represented by the sub-graph starting from it, as it can be seen on Fig. 5.1b. To emphasize the decision variables of the nodes, we mark them explicitly on the edges. Further, beyond the non-terminal (i.e. non-leaf) nodes we distinguish non-unit nodes that are representing a constraint over more than one variable. For example, the BDD of Fig. 5.1b contains two leaf nodes ($\top$ and $\bot$), two unit nodes (over x_3) and three non-unit non-leaf nodes. The ordering of the variables appearing in BDDs is fixed (e.g. $x_1 < x_2 < x_3$ in Fig. 5.1), i.e. we use ordered BDDs (OBDD in short). Even though we merge isomorphic subtrees in our BDDs, they are not reduced because nodes with identical children are kept (see e.g. x_3 in Fig. 5.1). Thus we use partially reduced ordered BDDs (ROBDD) over multiple variable orders.



(a) BDD of $(x_1 + x_2 + x_3 \leq 1)$ **(b)** Expanded BDD of $(x_1 + x_2 + x_3 \leq 1)$

Figure 5.1: Different BDD representations of AMO constraint $(x_1 + x_2 + x_3 \leq 1)$.

Given a graph $G = (V, E)$, a feasible solution to the antibandwidth problem consists of assigning each node $v \in V$ a unique label from the range $1, \ldots, |V|$. Given such a labeling f, the antibandwidth $AB_f(v)$ of a node v is defined as $\min\{|f(v) - f(v')| : \{v, v'\} \in E\}$, and the antibandwidth $AB_f(G)$ is defined as $\min\{AB_f(v) : v \in V\}$. The goal of the ABP is to find a labeling f^*, such that $f^* = \arg\max_{f \in \mathcal{F}(G)} AB_f(G)$, where $\mathcal{F}(G)$ denotes the set of all labelings of G.

We briefly discuss previous work [220] on which our new SAT solution is based. Let binary variables $x_i^\ell = 1$ if and only if vertex i is assigned label ℓ (i.e. $f_i = \ell$). For a given k, the question, whether there exists a solution with $\mathrm{AB}(G) \geq k + 1$, can be formulated as MIP as follows. We will denote this formulation as $F_e(k)$.

$$\max\ 0$$

$$\sum_{i \in V} x_i^\ell = 1 \qquad \forall \ell \in \{1, \ldots, |V|\} \qquad \text{(Labels)}$$

$$\sum_{\ell \in \{1,\ldots,|V|\}} x_i^\ell = 1 \qquad \forall i \in V \qquad \text{(Vertices)}$$

$$\sum_{\lambda \leq \ell \leq \lambda + k} (x_i^\ell + x_{i'}^\ell) \leq 1 \qquad \forall \{i, i'\} \in E,\ 1 \leq \lambda \leq |V| - k \qquad (\text{Obj}_k)$$

$$x_i^\ell \in \{0, 1\} \qquad \forall i \in V, \forall \ell \in \{1, \ldots, |V|\}$$

Constraints (LABELS) make sure that each label is used only once and constraints (VERTICES) ensure that each node $i \in V$ gets assigned one label. Thus, the solution encoded by these constraints corresponds to a labeling. Constraints (OBJ$_k$) describe that for each edge $\{i, i'\}$, the labels f_i, $f_{i'}$ are not allowed to be within a range of k. Thus, any solution of the above constraints corresponds to a labeling with antibandwidth at least $k + 1$. The iterative algorithm of [220] starts with a value of k obtained by a heuristic, which constructs a feasible labeling, and then iteratively solves $F_e(k)$ and increases k by one, until either $F_e(k)$ becomes infeasible (proving optimality of k) or a time limit is reached.

5.3 Staircase At-Most-One Constraint Sets

As a first step we define and illustrate the main concept of our paper, the so-called staircase AMO constraint set (SCAMO). Following that, in the next section we demonstrate step-by-step our proposed SAT encoding of these constraints.

Definition 5.3.1. *Given a sequence of Boolean variables $X = \langle x_1\, x_2 \cdots x_n \rangle$ and a width w s.t. $1 < w \leq n$, a* staircase constraint set *is formulated as follows:*

$$\text{SCAMO}(X, w) = \bigwedge_{i=0}^{(n-w)} \left(\sum_{j=i+1}^{(i+w)} x_j \leq 1 \right) \quad \textit{where } n = |X|.$$

Notice that this constraint is a special sub-case of SEQUENCE constraints (see e.g., [29, 33, 57, 229]) and so could be formulated as SEQUENCE$(0, 1, w, X, \{1\})$.

In Example 5.1.1 we saw, that there is an ordering of the constraints in that problem such that each constraint differs only slightly from the previous one. For instance, in Example 5.1.1 the 1st and 2nd constraints both include the sum of x_2, x_3 and x_4 while the 2nd and 3rd both contain the sub-expression $x_3 + x_4 + x_5$. Since addition is associative, the sum of the variables can be calculated regardless of the grouping of the variables. However, if we would like to reuse previous calculations, it is more beneficial to evaluate the first AMO constraint for example as $x_1 + (x_2 + x_3 + x_4)$ instead of considering any other variable grouping (e.g. $(x_1 + x_2) + (x_3 + x_4)$). Doing so, the second constraint can just simply consider the result of $(x_2 + x_3 + x_4)$ together with x_5. Continuing the evaluation with the next constraint, we could reuse $(x_3 + x_4)$ from $(x_2 + x_3 + x_4)$, in case we calculated it as $x_2 + (x_3 + x_4)$, to decide $x_3 + x_4 + x_5 + x_6 \leq 1$ by combining it with $(x_5 + x_6)$. In general, each constraint shares a sub-sum over $w - 1$ variables with the previous and at the same time with the next constraint.

Evaluating the very first constraint in this example in a right associative way allows us to reuse (at least once) all its sub-expression in the following three (i.e. $w - 1$) constraints. However, in order to reuse these sub-expressions we need a left associative grouping of variables in the constraint $x_5 + x_6 + x_7 + x_8 \leq 1$,

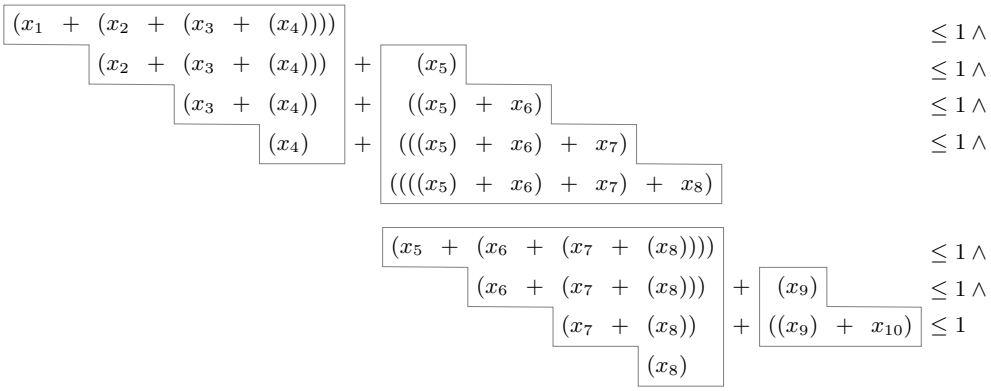


Figure 5.2: Decomposition of the staircase AMO constraint set of Example 5.1.1.

since in the second constraint we need x_5, then $(x_5 + x_6)$ and then $(x_5 + x_6 + x_7)$ to complement the reused sub-sums of $x_1 + x_2 + x_3 + x_4$.

All in all, considering only the first w constraints, we see that we need a right associative evaluation of the first constraint and a left associative grouping of the $(w+1)$'th constraint. Figure 5.2 depicts how these variable groupings can be "bonded" together to reconstruct the original constraints of Example 5.1.1. Extending this pattern to the whole set of constraints, we can see that each w consecutive constraints need to be considered once left associative to combine with the previous w constraints' sub-expressions and once right associative, to combine with the next w constraints. Thus, in Fig. 5.2 the sum over variables x_5, x_6, x_7 and x_8 is actually considered twice, once with a left and once with a right associative variable ordering. This duplicate view of constraints is the main concept behind our proposed duplex encoding.

5.4 Duplex Encoding of Staircase Constraint Sets

Our goal is to exploit sharing of sub-expressions between constraints to obtain a compact encoding. Again, the main idea of our approach can be seen in Fig. 5.2 where we identified common sub-sums. In our concrete encoding we have to go one step further though and actually have to share sub-constraints. This is achieved by decomposing longer AMO constraints into two smaller ones using the following proposition. While the original longer constraints may be used only once, smaller constraints potentially can be shared and reused multiple times.

Proposition 5.4.1. *A constraint* $x_1 + \cdots + x_n \leq 1$ *holds iff for all* $1 \leq i < n$

$(x_1 + \ldots + x_i \leq 1) \wedge (x_{i+1} + \ldots + x_n \leq 1) \wedge (x_1 + \ldots + x_i \leq 0 \vee x_{i+1} + \ldots + x_n \leq 0)$.

5.4.1 Sub-Constraint Construction

As a first step, given a sequence of variables $X = \langle x_1 \cdots x_n \rangle$ and width w, we partition the variables into $M = \lceil \frac{n}{w} \rceil$ consecutive *win-*

```
BDD-AMO (consecutive variables ⟨x_i ··· x_j⟩)
1 B := Search-AMO(⟨x_i ··· x_j⟩)
2 if B = ∅ then
3     if |⟨x_i ··· x_j⟩| = 1 then
4         B_T, B_F := ⊤, ⊤
5     else
6         B_T := BDD-AMZ(⟨x_{i+1} ··· x_j⟩)
7         B_F := BDD-AMO(⟨x_{i+1} ··· x_j⟩)
8     B := if-then-else(x_i, B_T, B_F)
9 return B
```

```
BDD-AMZ (consecutive variables ⟨x_i ··· x_j⟩)
1 B := Search-AMZ(⟨x_i ··· x_j⟩)
2 if B = ∅ then
3     if |⟨x_i ··· x_j⟩| = 1 then
4         B_T, B_F := ⊥, ⊤
5     else
6         B_T := ⊥
7         B_F := BDD-AMZ(⟨x_{i+1} ··· x_j⟩)
8     B := if-then-else(x_i, B_T, B_F)
9 return B
```

Figure 5.3: Algorithms BDD-AMO and BDD-AMZ to construct binary decision diagrams for constraints over a given sequence of consecutive Boolean variables.

dows $\omega_1, \omega_2, \ldots, \omega_M$, where ω_1 contains variables $x_1, \ldots, x_w$, ω_2 contains $x_{w+1}, \ldots, x_{2w}$ etc. Note that unless $(n \bmod w) = 0$, the very last window contains fewer than w variables.

Example 5.4.1. *Continuing the previous example, our width* $w = 4$ *splits* X *into three windows:* $\omega_1 = \{x_1, x_2, x_3, x_4\}$, $\omega_2 = \{x_5, x_6, x_7, x_8\}$ *and* $\omega_3 = \{x_9, x_{10}\}$.

To encode a SCAMO set of constraints as compositions of smaller constraints, we build two BDDs for each window with two different variable orderings (hence the name "duplex"). Notice that any SAT encoding technique of AMO constraints could be employed instead of BDDs (as long as we do duplex encoding by considering both directions). However, beyond the smaller AMO constraints, we further need AMZ constraints in order to connect the parts together (see the binary clause in Prop. 5.4.1). One benefit of BDDs is that we get these constraints automatically already by encoding the AMO constraints. Thus in this paper we will focus only on this BDD based approach.

Given window ω_i over variables $X_i = \{x_{i_1}, \ldots x_{i_w}\}$, we construct two two-rooted BDDs, both representing the same two constraints $x_{i_1} + \cdots + x_{i_w} \leq 1$ and $x_{i_1} + \cdots + x_{i_w} \leq 0$. The first BDD, which we call *forward* BDD, considers the AMO and AMZ constraints with a right associative variable grouping (i.e. with variable ordering $x_{i_1} < x_{i_2} < \ldots < x_{i_w}$). The other BDD, called *backward* BDD, represents the same constraints but with a left associative variable grouping (i.e. with variable ordering $x_{i_w} < x_{i_{w-1}} < \ldots < x_{i_1}$).

Abío et al. in [2] proposed a generalized arc-consistent, polynomial size ROBDD-based encoding for Pseudo-Boolean constraints. In our setting the constraints are all AMO or AMZ constraints without coefficients, and thus applying their approach leads to small and simple BDDs. The recursive algorithms in Fig. 5.3 present the main steps of this building process. In these procedures $\langle x_i \cdots x_j \rangle$ means an ordered sequence of consecutive variables and function `if-then-else` builds a BDD node with the given decision variable and high and low BDD nodes. Building the forward BDDs of a window ω_i simply means to call BDD-AMO and BDD-AMZ with $\langle x_{i_1} \cdots x_{i_w} \rangle$ as parameter. To build

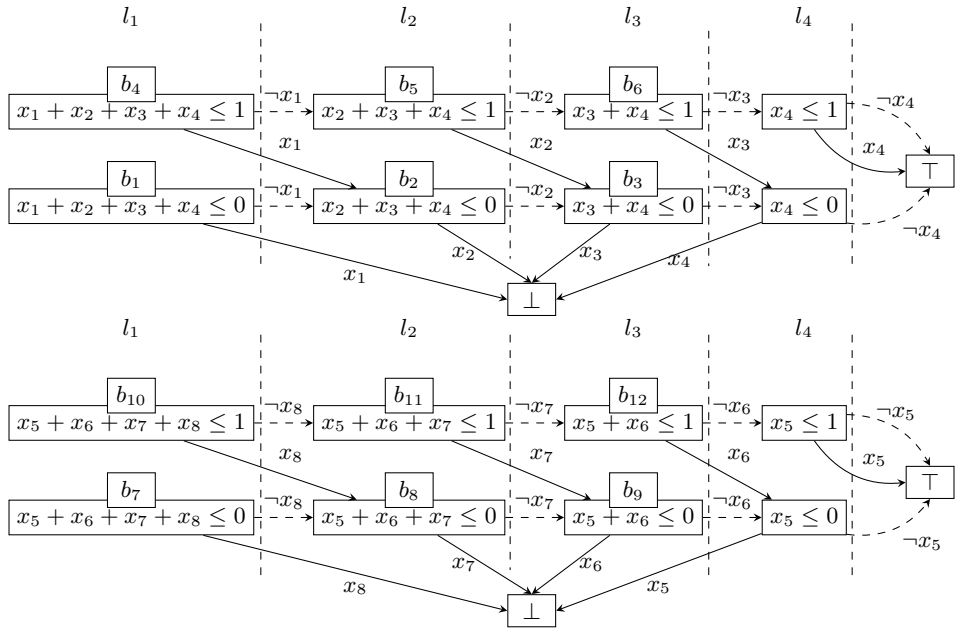


Figure 5.4: Forward BDD of ω_1 with variable ordering $x_1 < x_2 < x_3 < x_4$ and backward BDD of ω_2 with ordering $x_8 < x_7 < x_6 < x_5$. Two-rooted partially reduced OBDDs to represent constraints $x_1+x_2+x_3+x_4 \leq K$ with right and $x_5+x_6+x_7+x_8 \leq K$ with left associative variable groupings, where $K \in \{0, 1\}$.

the backward BDDs, we need to call the methods with $\langle x_{i_w} \cdots x_{i_1} \rangle$ as argument. The result in both cases (see Ex. 5.4.2) will be a two-rooted BDD with height of at most $(w + 1)$.

Consider the following *layers* of these constructed BDDs. A non-leaf layer l_j (where $1 \leq j \leq w$) of a forward BDD (backward BDD) consists of two nodes, one capturing the AMO and another node representing the AMZ constraint over variables $\langle x_{i_j} \cdots x_{i_w} \rangle$ (respectively $\langle x_{i_{w-(j-1)}} \cdots x_{i_1} \rangle$ for the backward BDD).

Example 5.4.2. *The upper part of Fig. 5.4 shows what the forward BDD of ω_1 in Example 5.4.1 looks like. The BDD is the result of calling* `BDD-AMO`*($\langle x_1\, x_2\, x_3\, x_4 \rangle$) and* `BDD-AMZ`*($\langle x_1\, x_2\, x_3\, x_4 \rangle$). Notice that due to the search for already existing BDDs at the beginning of each method (*`Search-AMO` *and* `Search-AMZ`*), the two calls result in a single shared structure (i.e. we have a partially reduced ordered BDD). Further notice that though node $x_4 \leq 1$ could be reduced simply to $\top$, we kept this node in the representation. In this BDD we can distuinguish four layers ($l_1 - l_4$) that refer to four sub-constraints of the root expressions.*

The lower part of the figure depicts the backward BDD of ω_2 in Example 5.4.1, resulting from calls `BDD-AMO`*($\langle x_8\, x_7\, x_6\, x_5 \rangle$) and* `BDD-AMZ`*($\langle x_8\, x_7\, x_6\, x_5 \rangle$). The variable ordering here is $x_8 < x_7 < x_6 < x_5$. Notice that the structure of the two BDDs are identical, they just talk about different variables in different orders.*

5.4.2 CNF Encoding of BDDs

During BDD construction (e.g. after Line 5 in both algorithms of Fig. 5.3), or later in an independent traversal, we can assign new Boolean variables to each non-unit non-leaf node. Notice that top nodes of the forward and backward BDDs over the same variables can use the same Boolean variable.

Now, given a node with auxiliary Boolean variable b, that decides on variable x_i and has a true child node with variable t and a false child node with variable f, we introduce clauses to encode $x_i \to (b \leftrightarrow t)$ and $\neg x_i \to (b \leftrightarrow f)$. However, there are several simplification possibilities due to the structure of our BDDs and our problem. For instance, all AMZ nodes have $\bot$ as a true child (see Fig. 5.4) and all AMO nodes are assumed as unit clauses (due to using them with Prop. 5.4.1). Nodes of a constraint $x_i \leq 1$ are simply encoded as $\top$, while nodes of constraints $x_i \leq 0$ are encoded as $\neg x_i$ in the clausal representation of the parent nodes.

Example 5.4.3. *On Fig. 5.4 the introduced new Boolean variables are represented together with their nodes. For example, variable b_6 belongs to the node of constraint $x_3 + x_4 \leq 1$. The introduced clause regarding this node is $(\neg x_3 \vee \neg x_4)$.*

5.4.3 Bonding Stairs

An AMO constraint of a SCAMO set is either a root node of one of our BDDs or can be described by combining two layers of two BDDs via Prop. 5.4.1. As last step of encoding a whole SCAMO set of constraints, we traverse the forward BDD of each window (denoted as ω_i^f-BDD with $i \in \{1, \ldots, M-1\}$) and combine its nodes with those of the backward BDD of the next window (ω_{i+1}^b-BDD). Thus, we combine layer l_j of ω_i^f with layer $l_{(w-j)+2}$ of ω_{i+1}^b for each $j = 2, \ldots, w$. At the end, the bonding of two consecutive BDDs yields the following formula:

$$\mathrm{BOND}(\omega_i^f, \omega_{i+1}^b) \;=\; \omega_i^f\text{-}l_1\text{-AMO} \wedge \bigwedge_{j=2}^{w} \Big(\omega_i^f\text{-}l_j\text{-AMO} \wedge \omega_{i+1}^b\text{-}l_{(w-j)+2}\text{-AMO} \wedge (\omega_i^f\text{-}l_j\text{-AMZ} \vee \omega_{i+1}^b\text{-}l_{(w-j)+2}\text{-AMZ})\Big).$$

Example 5.4.4. *We continue the running example. At this point we have seen how to construct a BDD for each small stair structure in Fig. 5.2. Next we combine them using Prop. 5.4.1 to capture all AMO constraints. Fig. 5.5 depicts how the layers of the constructed BDDs are meant to be paired with each other. Applying Prop. 5.4.1 on layers of ω_1^f-BDD and ω_2^b-BDD yields the following formula:*

$$\begin{gathered}(x_1 + x_2 + x_3 + x_4 \leq 1) \wedge \\ (x_2 + x_3 + x_4 \leq 1) \wedge (x_5 \leq 1) \wedge (x_2 + x_3 + x_4 \leq 0 \vee x_5 \leq 0) \wedge \\ (x_3 + x_4 \leq 1) \wedge (x_5 + x_6 \leq 1) \wedge ((x_3 + x_4 \leq 0) \vee (x_5 + x_6 \leq 0)) \wedge \\ (x_4 \leq 1) \wedge (x_5 + x_6 + x_7 \leq 1) \wedge ((x_4 \leq 0) \vee (x_5 + x_6 + x_7 \leq 0)),\end{gathered}$$

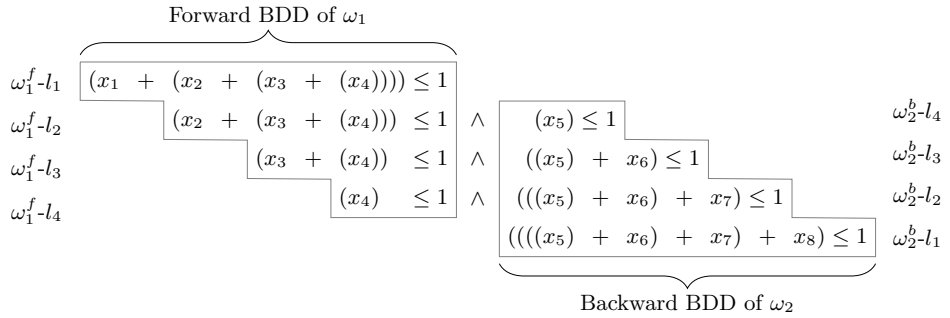


Figure 5.5: Combining forward and backward BDDs to encode SCAMO constraints.

that translates to the clauses $b_4 \wedge b_5 \wedge \top \wedge (b_2 \vee \neg x_5) \wedge b_6 \wedge b_{12} \wedge (b_3 \vee b_9) \wedge \top \wedge b_{11} \wedge (\neg x_4 \vee b_8)$*. Notice that with this set of clauses, together with the BDD clauses, we encoded the first four AMO constraints of our SCAMO problem.*

5.4.4 Arc Consistency of Duplex Encoding

Notice that AMO, AMZ and SCAMO constraints are all monotonic decreasing Boolean functions, i.e. setting any of the variables to false does not restrict any other variables. Thus setting a variable to true affects only those variables that share at least one AMO constraint with it. Note that decomposing each AMO constraint of a SCAMO set based on Prop. 5.4.1 results in an equivalent problem. Although our constructed BDDs for this decomposition share most of their nodes with each other (due to the chosen variable orders), our method is still a BDD-based translation of each AMO and AMZ constraint into clauses. Thus, applying an arc consistent encoding [14, 104] on each BDD node (e.g. the one in Minisat+ [90]) makes our encoding arc consistent as well.

In fact, notice that our bonding clauses contain a unit clause for each AMO constraint in order to enforce the output of the corresponding (sub-)BDD to be true. Beyond that, it is not hard to see that setting an input variable to true falsifies the output variable of each AMZ-BDD containing it. Thus the binary clauses of the bonding clauses enforce the root-node of each respective AMZ constraint to be true, and in turn unit propagation, the main inference rule of SAT solvers, falsifies all the variables in them.

5.5 Comparing Encodings of Staircase Constraints

In this section we discuss commonly used existing SAT encodings of AMO constraints and possible SEQUENCE encodings of SCAMO constraints. We compare them to our proposed duplex encoding in the context of SCAMOs.

Let $N = (n-w)+1$ be the number of AMO constraints in a staircase problem set over n variables and width w. A *naive* (also called *pair-wise* or *binomial*) encoding of a w-long AMO constraint is $\bigwedge_{i=1}^{(w-1)} \bigwedge_{j=i+1}^{(w)} (\neg x_i \vee \neg x_j)$. Although this approach does not require any additional Boolean variable, the number

of clauses constructed with that encoding over N w-long AMO constraints is $N \cdot ((w-1) + (w-2) + \ldots + (w-(w-1))) = N \cdot \frac{(w-1)\cdot w}{2}$.

Using the naive encoding on the SCAMO constraint set would produce more than once many of the binary clauses. Eliminating duplicated clauses yields the *reduced naive* encoding with $\frac{(w-1)\cdot w}{2} + (N-1)\cdot(w-1)$ unique clauses.

Sinz introduced in [221] a *sequential* counter encoding for Boolean cardinality constraints. Applying it to an AMO constraint over w variables produces $3 \cdot w - 5$ binary clauses and introduces $w-2$ auxiliary variables. With N AMO constraints this gives $N \cdot (3 \cdot w - 5)$ clauses and $N \cdot (w-2)$ new variables.

The *BDD-based* encoding for Pseudo-Boolean constraints [2, 90] applied to AMO constraints is comparable to the sequential counter encoding. However, for a fixed variable order, the BDD built for each w-long AMO constraint of a SCAMO set, will always either contain a variable that does not occur in any other constraint or will miss a variable needed in other constraints. Thus for this approach using a fixed single variable order the amount of sharing of BDD nodes among constraints is rather restricted. On the other hand the approach does not require bonding clauses. With a simplified clausal representation of the BDD nodes, the naive BDD encoding uses at most $N \cdot (3 \cdot (w-2) + 2 \cdot (w-1) - 1)$ clauses and introduces $N \cdot (2 \cdot w - 3)$ new variables to encode a SCAMO set.

The so-called *2-product* encoding [70] relies on the same decomposition rule as Prop. 5.4.1. This approach breaks an AMO constraint over w variables into a product of two AMO constraints over p and q variables, where $p * q \geq w$. To simplify the calculation we use $p = \lceil \sqrt{w} \rceil$ and $q = \lceil w/p \rceil$ as recommended in [70] and assume recursive 2-product encoding of the resulting smaller constraints. Even though this approach can efficiently encode a single AMO constraint, making use of shared sub-expressions is not straightforward. Thus, based on the estimations given in [70], the number of clauses is $N \cdot (2 \cdot w + 4 \cdot \sqrt{w} + O(\sqrt[4]{w}))$. Further, the number of newly introduced variables is $N \cdot (2 \cdot \sqrt{w} + O(\sqrt[4]{w}))$ again following [70].

Instead of focusing on specialized AMO encodings, it is also possible to encode a complete SCAMO set with more generic approaches, like the ones in [57]. For example, encoding SCAMO as a REGULAR constraint yields similar results as a naive BDD-based approach with a single variable order (i.e. $\mathcal{O}(n \cdot w)$ size).

Another encoding (also from [57]) based on cumulative sums or difference constraints requires an internal representation which is at least quadratic size in the worst case. Similarly, partial sums (again see [57]) would consider every possible sub-sums which also yields $\mathcal{O}(n \cdot w^2)$ constraints.

The size-wise most competitive sequence encoding from [57] is the log-based approach where a SCAMO set could be represented as $\mathcal{O}(n \cdot log\, w)$ constraints.

5.5.1 Duplex Encoding

For a given constraint set over n variables of width w we construct two BDDs of the same size (each having $2 \cdot (w+1)$ nodes) for $M = \lceil \frac{n}{w} \rceil$ windows. To simplify

Table 5.1: Comparison of size of SAT encodings of w-long SCAMO sets over n variables. Columns #NewVars and #Clauses show the number of additional variables and clauses of each approach, where $N = (n - w) + 1$ and $M = \lceil \frac{n}{w} \rceil$.

Encoding	#NewVars	#Clauses	WorstCase
Naive	0	$N \cdot \frac{(w-1)\cdot w}{2}$	$\mathcal{O}(n^3)$
Reduced	0	$\frac{(w-1)\cdot w}{2} + (N-1)\cdot(w-1)$	$\mathcal{O}(n^2)$
Sequential	$N \cdot (w-2)$	$N \cdot (3 \cdot (w-2)+1)$	$\mathcal{O}(n^2)$
BDD	$N \cdot (2 \cdot w - 3)$	$N \cdot (3 \cdot (w-2) + 2 \cdot (w-1) - 1)$	$\mathcal{O}(n^2)$
2-Product	$N \cdot (2 \cdot \sqrt{w} + O(\sqrt[4]{w}))$	$N \cdot (2 \cdot w + 4 \cdot \sqrt{w} + O(\sqrt[4]{w}))$	$\mathcal{O}(n^2)$
Duplex	$4 \cdot M \cdot (w-1)$	$13 \cdot M \cdot w - 14 \cdot M - 3 \cdot w + 2$	$\mathcal{O}(n)$

the calculation, we will assume that each BDD has the same size (even though the last window is most of the time way smaller) and that we encode the first and last windows in both directions. Thus, we provide here just an upper bound on the actual values. With these assumptions we have $2 \cdot M$ BDDs. For each BDD we construct three clauses for the non-unit non-leaf AMZ nodes and at most two clauses for the non-unit non-leaf AMO nodes. Beyond these clauses, we need to bond together each layer of the neighbouring forward and backward BDDs, resulting in $M - 1$ bond-clause sets, each consisting of two unit and a binary clause. All in all, the final number of clauses in the encoding is as follows:

$$\#\text{BDD-clauses} \leq 2 \cdot M \cdot (3 \cdot (w-1) + 2 \cdot (w-1) - 1) = 10 \cdot M \cdot w - 12 \cdot M$$
$$\#\text{BOND-clauses} \leq (M-1) \cdot (3 \cdot (w-1) + 1) = 3 \cdot M \cdot w - 2 \cdot M - 3 \cdot w + 2$$
$$\#\text{BDD} + \#\text{BOND-clauses} \leq 13 \cdot M \cdot w - 14 \cdot M - 3 \cdot w + 2$$

The number of new variables at the very end of the encoding is at most $4 \cdot M \cdot (w-1)$ introducing one for each non-leaf non-unit node of our BDDs.

5.5.2 Comparison Summary

Table 5.1 summarizes the sizes of different SAT encodings expressed as functions over the number n of all variables in a SCAMO constraint set and the width w of the individual AMO constraints, combined into $N = (n - w) + 1$ (the number of AMO constraints) and $M = \lceil \frac{n}{w} \rceil$ (the number of windows in duplex encoding). The columns capture the number of auxiliary variables and number of clauses of the encodings. Notice that M is significantly smaller than N. The last column gives the worst case of each approach, assuming $w = n/2$, where N is approximately $n/2$ too. In this scenario existing encodings are quadratic or even cubic. However, in our duplex encoding we have $M = 2$ in that case and thus it remains linear.

Figure 5.6 visualizes the difference between SAT encodings for the fixed number of variables $n = 500$. The horizontal axis ranges over all possible widths w. Note that the naive encoding is only partially shown here, and further, that in our application $n/2$ is an upper bound on the width w, and thus only the left part of Fig. 5.6 is interesting up to the middle $w = n/2 = 250$.

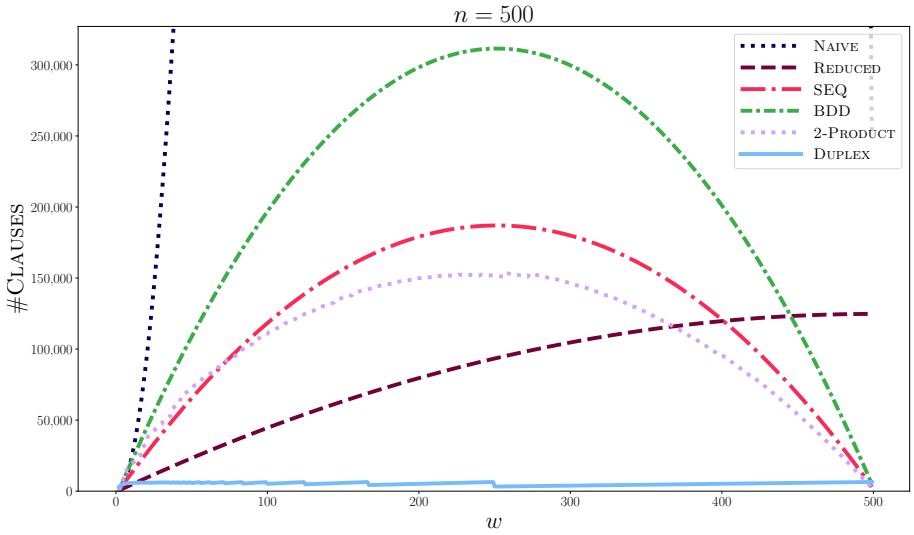


Figure 5.6: Comparison of number of clauses for different encodings of a single SCAMO constraint set on $n = 500$ variables and width w between 2 and 500.

The asymptotic behavior of the last column of Table 5.1 can be observed in Fig. 5.6 too. Again, the largest difference between the encodings occurs for $w = n/2$. According to Fig. 5.6 the reduced naive encoding turns out to be the best SAT-based alternative to our approach in terms of number of clauses. Though Fig. 5.6 focuses only on SAT encodings, note that the smallest sequence-based alternative (in [57]) would have size $\mathcal{O}(n \cdot log\, n)$ when $w = n/2$, that is smaller than most SAT encodings but larger than our proposed linear encoding.

5.6 Experimental Evaluation

Formulating the antibandwidth problem iteratively, as it was proposed in [220] (see Sect. 5.2), asks whether there exists a labelling for a graph $G = (V, E)$ s.t. $\mathrm{AB}(G) \geq k + 1$. The question has $2 \cdot |V|$ pieces of $|V|$-long exactly-one constraints (as (Labels) and (Vertices)) and for each edge of the graph (i.e. $|E|$ times) a $(|V| - k)$ big set of AMO constraints, each over $2 \cdot k$ variables (as (Obj_k)).

An off-the-shelf SAT solution could encode each of the AMO and exactly-one constraints one-by-one (e.g. as in Sect. 5.5). However, for a given edge between nodes i, i' (i.e. $\{i, i'\} \in E$) constraint (Obj_k) can be reformulated as

$$\bigwedge_{\lambda=1}^{(|V|-k)} \left(\sum_{\ell=\lambda}^{(\lambda+k)} x_i^\ell + x_{i'}^\ell \leq 1 \right) \overset{\textit{Prop. } 5.4.1}{\equiv}$$

$$\bigwedge_{\lambda=1}^{(|V|-k)} \left(\sum_{\ell=\lambda}^{(\lambda+k)} x_i^\ell \leq 1 \wedge \sum_{\ell=\lambda}^{(\lambda+k)} x_{i'}^\ell \leq 1 \wedge \left(\sum_{\ell=\lambda}^{(\lambda+k)} x_i^\ell \leq 0 \vee \sum_{\ell=\lambda}^{(\lambda+k)} x_{i'}^\ell \leq 0 \right) \right).$$

In that form we have exactly two SCAMO sets of width $k+1$, one over the variables of node i and another over variables of i'. The third component of the decomposition takes the disjunction of AMZ constraints that can be constructed easily by combining our smaller AMZ nodes corresponding to the SCAMO sets.

The staircase structure in (OBJ_k) allows to apply our new duplex encoding by simply encoding a SCAMO set of width $k+1$ for each node of the graph (i.e. $|V|$ times) and combining the corresponding AMZ constraints (with less than $4 \cdot (|V|-k)$ binary clauses for each edge). This encodes all AMO constraints of the problem. Also note that we can reuse the Boolean variables representing the root nodes of the constructed AMO BDDs to encode the (VERTICES) constraints.

Experimental Results

We implemented a framework to compare off-the-shelf SAT encodings in practice to our proposed SCAMO based duplex encoding on the antibandwidth problem (as formulated in Sect. 5.2). Beyond SAT encodings, we also compared our approach against alternative exact methods to solve the problem, like Constraint Programming or the iterative method presented in [220] based on feasibility-MIPs.

The experiments considered 24 matrices of the Harwell-Boeing Sparse Matrix Collection [208], containing 12 relatively small and 12 rather large graphs (as in [220]). For each graph lower bounds (by a construction heuristic) and theoretical upper bounds of the antibandwidth were provided in [220]. These values were reused in our experiment as starting and ending points for the iterative methods and as lower bounds in the CP approaches. All reported results were experimented on our cluster with Intel Xeon E5-2620 v4 @ 2.10GHz CPUs.

Table 5.2 summarizes our results.[1] For each graph it shows the number of nodes and edges, the starting width or lower bound and last width to check of the solving methods (columns $|V|, |E|$ and LB,UB). Then for each solving technique we report the best found solution together with the time (in seconds) and memory consumption (in MB). Each approach was limited to 1800 seconds and 120 GB memory. This rather high main memory limit is due to trying to solve the alternative SAT encodings with a large number of clauses as well, while the other methods never exceeded 4 GB.

We compare the 2-product [70] and reduced naive AMO encodings to our proposed duplex SCAMO encoding as the first three techniques in Table 5.2. All three techniques are implemented in the same framework and follow the same method: encode (considering LB as width of SCAMO or as k of the AMO constraints) and solve the SAT representation of the problem with a SAT solver (we used CaDiCaL 1.2.1 [38]). If it is satisfiable, increase the width and start again to encode and solve the new problem. If it is unsatisfiable or the width is UB, it means that the optimal solution of ABP was found and the process ends.

[1] Source code, data and benchmarks are available at `http://fmv.jku.at/duplex/`.

At the moment when the 1800 seconds or 120 GB is exceeded, the method stops (with TO or MO respectively). The reported solutions are the highest widths with what the formula was still successfully constructed and solved. In case even the first formula was too hard to solve, it is marked with "-".

While the 2-product encoding of the largest instance had a memory out during solving the first formula (after a successful encoding), the reduced naive approach required less memory and even solved a few of the larger problems with more than one width in 1800 seconds. The duplex encoding required significantly less memory and was faster in encoding and solving the problems compared to the other SAT approaches. It performed well also compared to further techniques.

The next two approaches, $F_e(k)$ and CP-CPLEX, are taken from [220] as is, and were executed on our cluster for comparison. Note that while CP-CPLEX knows LB, $F_e(k)$ constructs it internally. The last reported approach is based on Chuffed [98, 226] via the MiniZinc language [188]. This hybrid solver employs lazy clause generation and combines the strengths of SAT and finite domain solving techniques. Note that both CP approaches encode the ABP naively as a labeling problem to maximize smallest neighbour-distances, using state-of-the-art solvers off-the-shelf. All in all we can see that the SCAMO based duplex encoding of the ABP is comparable and most of the time even better than other approaches.

5.7 Conclusion and Outlook

In this paper we have proposed a new SAT encoding for at-most-one constraints with a staircase structure, i.e. where consecutive constraints share sequences of sub-expressions in a structured way. This structure is exploited in an encoding which relies on binary decision diagrams using two variable orderings. Compared to alternative encodings for the ABP, our encoding outperforms the existing ones.

In the future we plan to integrate and interleave the MIP based approach of [220] and the SAT approach proposed here. Further, we want to apply the proposed method to other problems featuring at-most-one constraints with a staircase structure. Another intriguing direction for future work is to explore how symbolic optimization techniques using decision diagrams [32] can take advantage of multiple variable orders simultaneously, which is essential to keep our encoding compact.

Acknowledgments.

This research has been supported by the Austrian Science Fund (FWF) under projects W1255-N23, S11408-N23 and by the LIT AI Lab funded by the State of Upper Austria. The authors would like to thank the reviewers for their useful suggestions and helpful comments.

Table 5.2: Results of different approaches to solve the antibandwidth problem (TO = 1800 seconds and MO = 120 GB).

Instance	$\|V\|$	$\|E\|$	LB	UB	2-Product			Reduced Naive			Duplex			F_e(k) [220]			CP-CPLEX [220]			CP-MZ-Chuffed		
					Obj.	Time	MB	Obj.	Time	MB	Obj.	Time	MB	Obj.	Time	MB	Obj.	Time	MB	Obj.	Time	MB
A-pores_1	30	103	6	8	**6**	206.85	80	**6**	166.48	68	**6**	185.52	52	**6**	23.71	29	6-8	TO	57	**6**	5.97	11
B-ibm32	32	90	9	9	**9**	14.06	51	**9**	46.03	47	**9**	1.30	11	**9**	28.57	29	**9**	7.35	20	**9**	17.4	11
C-bcspwr01	39	46	16	17	**17**	83.12	69	**17**	56.02	59	**17**	3.85	13	**17**	6.64	28	**17**	18.78	21	17	TO	11
D-bcsstk01	48	176	8	9	**9**	14.41	139	**9**	8.59	47	**9**	0.25	14	**9**	62.28	36	**9**	20.15	21	**9**	6.35	12
E-bcspwr02	49	59	21	22	**21**	36.17	76	**21**	53.01	80	**21**	3.37	13	**21**	774.02	205	**21**	22.84	19	**21**	673.44	11
F-curtis54	54	124	12	13	**13**	20.89	139	**13**	1.02	41	**13**	1.33	18	**13**	12.56	32	**13**	34.66	21	**13**	2.14	11
G-will57	57	127	12	14	**13**	108.79	164	**13**	26.8	79	**13**	0.57	19	**13**	15.4	33	**13**	44.75	21	**13**	2.69	11
H-impcol_b	59	281	8	8	**8**	5.51	173	**8**	0.47	52	**8**	0.54	22	**8**	0.47	24	8-22	TO	63	**8**	23.3	12
I-ash85	85	219	19	27	21	TO	794	21	TO	658	**23**	TO	331	20	TO	133	22-31	TO	37	21	TO	12
J-nos4	100	247	32	40	32	TO	1037	32	TO	911	**35**	585.33	190	-	TO	106	34-47	TO	31	-	TO	12
K-dwt_234	117	162	46	58	47	TO	924	47	TO	957	49	TO	477	48	TO	264	**51-57**	TO	33	-	TO	11
L-bcspwr03	118	179	39	39	**39**	22.82	662	**39**	6.92	436	**39**	0.99	58	**39**	0.52	21	**39**	110.92	22	**39**	26.42	12
M-bcsstk06	420	3720	28	72	-	TO	53392	29	TO	22076	**34**	TO	1621	33	TO	625	-	TO	20	-	TO	35
N-bcsstk07	420	3720	28	72	-	TO	53392	29	TO	22097	**34**	TO	1621	33	TO	634	-	TO	20	-	TO	35
O-impcol_d	425	1267	91	173	-	TO	30306	92	TO	22285	**99**	TO	1043	95	TO	691	-	TO	18	-	TO	24
P-can_445	445	1682	78	120	-	TO	41572	-	TO	27030	-	TO	1581	-	TO	644	-	TO	18	-	TO	24
Q-494_bus	494	586	219	246	-	TO	25944	-	TO	29640	-	TO	1167	**220**	TO	905	-	TO	18	-	TO	21
R-dwt_503	503	2762	46	71	-	TO	56611	47	TO	35227	**62**	TO	1680	52	TO	911	-	TO	19	-	TO	31
S-sherman4	546	1341	256	272	-	TO	73031	-	TO	59860	-	TO	1129	-	TO	1033	-	TO	19	-	TO	24
T-dwt_592	592	2256	103	150	-	TO	85816	-	TO	62936	-	TO	2253	-	TO	1068	-	TO	20	-	TO	37
U-662_bus	662	906	219	220	-	TO	63844	-	TO	68402	**220**	319.73	1564	-	TO	1320	-	TO	19	-	TO	28
V-nos6	675	1290	326	337	-	TO	101724	-	TO	90129	-	TO	1571	-	TO	1434	-	TO	20	-	TO	28
W-685_bus	685	1282	136	136	-	TO	76110	-	TO	72839	**136**	14.33	1428	**136**	9.24	37	-	TO	20	-	TO	29
X-can_715	715	2975	112	142	-	686.23	MO	-	TO	106462	-	TO	3312	-	TO	1468	-	TO	21	-	TO	39

Chapter 6

Extensions to Published Work

This chapter extends the peer-reviewed and published works presented in Chapter 2-5 with further details and results. Regarding Chapter 2, the extension is substantial, consisting of a preliminary version of a yet unpublished line of work (see Section 6.1). Regarding Chapter 3 and 5, the extensions in Section 6.2 and 6.4 are for the sake of completeness, presenting technical proofs and some experimental results that did not fit into the page limits of the proceedings. About incremental inprocessing in the context of SAT solving, beyond the published work in Chapter 4, some further refinements and theoretical results are presented in Section 6.3.

6.1 Quantified Boolean Formulas and Theory Reasoning

The long-term motivation behind our paper in Chapter 2, beyond presenting an abstract description of search-based QBF solvers, was to pave the path towards a framework where we can formally introduce theory reasoning into the process of QBF evaluation. Though this final goal has not been achieved yet, there are some interesting questions and partial results that did arise along the way and that are shortly described in the following.

6.1.1 Motivational Example

The following sentence, originating from synthesis problems as we will see later, illustrates the formulas that we address here. Consider the quantified formula

$$\begin{aligned}\mathcal{Q}.\mathcal{F} := \exists i\ \forall c\ \exists o\ .\ & P(i) \oplus P(f(i)) \wedge \\ & (c \Rightarrow o = i) \wedge (\neg c \Rightarrow o = f(i)) \wedge \\ & P(o)\end{aligned}$$

over the three variables i, o and c. The existentially quantified variables i and o in the formula prefix $\mathcal{Q}$ are representing input and output in this example and they are from the same arbitrary, finite domain (e.g. small integers or just uninterpreted). The universally quantified variable c represents control and it is Boolean. In the formula matrix $\mathcal{F}$ predicates and functions (P and f)

are applied, but their precise meaning is currently not relevant (i.e. they are uninterpreted, but for the sake of our example, P could be true for each even number while f could increase the value of a variable by one). The formula means that predicate P holds exclusively either on the input i directly, or on $f(i)$, while based on the truth value of c the output is equal either with i or with $f(i)$ and predicate P holds on the output. Notice that the formula is false, since for the same input one of the two possible c values always makes $P(o)$ false.

Although $\mathcal{Q}.\mathcal{F}$ is an overly simplified toy example, it actually captures the form of formulas that can occur in synthesis of conditions in loop-free programs or of control logic signals in hardware design. In microprocessor design and development, *pipelining* is a common way to enhance the throughput of the system. Burch and Dill in [63] described how to formally specify and verify the correctness of pipeline controllers. The main idea of their approach is to specify the externally visible behaviour of a non-pipelined reference design and a pipelined design, and compare the two designs to each other. Hofferek and Bloem [127,128] showed how to extend and adapt a Burch-Dill style specification of pipelined circuits to a synthesis setting, using uninterpreted functions. They introduced Boolean variables to represent the yet unknown control signals of the system, and with quantifier alternations maintained the dependencies between the states of the system and those control signals. The general meaning of their constructed formulas can be summarized with the following sentence: *"... for every possible state of the system, there exist values for the control signals, such that for all values of auxiliary variables (required to formulate the correctness criterion), the correctness criterion holds."*(see [127], Section 1.2, p.5). More formally, their specifications have the form $\forall \vec{x} \,.\, \exists \vec{c} \,.\, \forall \vec{x}' \,.\, \Phi$, where the variables from the first quantifier block (vector of variables noted as $\vec{x}$) describe the state and/or inputs of the system, $\vec{c}$ is a vector of propositional variables representing the control signals that are supposed to be synthesized, and the last quantifier block contains the auxiliary variables necessary to formulate the correctness criterion. The domain of $\vec{x}$ and $\vec{x}'$ is uninterpreted, while the variables in $\vec{c}$ are all Boolean. The matrix of the formula is in quantifier-free first-order logic, containing equalities over uninterpreted functions and any elements of the array property fragment of the theory of arrays (for details on this fragment see e.g. [53,54]). The generalized version of these formulas even allows an arbitrary number of quantifier alternations, as long as all the existential quantifiers are over Boolean variables only. Using that formalization, the synthesis problem reduces to the challenge of finding witness functions (*certificates*) that compute the values of the Boolean variables from the values of the variables in $\vec{x}$.

Our toy example $\mathcal{Q}.\mathcal{F}$ captures the shape (i.e. quantifier prefix and variable domains) of the negation of these simpler specification formulas. To further simplify the presentation we assume only uninterpreted functions and equalities in the formula matrix. This simplification is reasonable, since in practice array properties are usually reduced into that fragment of logic (together with the element and index theories).

6.1.2 Existing Solution Approaches

The first question is how to determine automatically the truth value of a formula like our motivational example. Without claiming to be exhaustive, here we briefly collect some procedures that could be used for this purpose.

Our formula $\mathcal{Q}.\mathcal{F}$ is definitely in first-order logic, there are quantifiers, predicate symbols, function symbols and equalities. With minor modifications one could give it to a first-order theorem prover (e.g. [148, 233]). However, theorem provers are not guaranteed to terminate since first-order logic is in general undecidable [72]. Further, the formula is relatively simple considering the complete expressiveness of first-order logic and thus probably a more specific approach could be more beneficial here.

Taking a closer look on the symbols in $\mathcal{F}$ one can see that only equalities and uninterpreted predicates and functions are used. Equalities over uninterpreted functions is actually an efficiently decidable background theory supported by most SMT solvers [193]. However, reasoning about quantified formulas in general is yet rather challenging for most SMT solvers, though several complete and incomplete instantiation techniques exist already [24,101,102,206]. A simple way to overcome this difficulty is to eliminate (i.e. expand) the universally quantified Boolean variable from the formula by considering both possible instantiations of it, as in the following transformation:

$$\begin{aligned}\mathcal{Q}.\mathcal{F} \equiv \exists i.\ & (\exists o_0.\ P(i) \oplus P(f(i)) \wedge o_0 = i \wedge P(o_0)) \wedge \\ & (\exists o_1.\ P(i) \oplus P(f(i)) \wedge o_1 = f(i) \wedge P(o_1)) \\ \equiv_{sat} & P(i) \oplus P(f(i)) \wedge o_0 = i \wedge P(o_0) \wedge o_1 = f(i) \wedge P(o_1).\end{aligned}$$

The final formula then can be solved easily with most SMT solvers. Using this transformation was found to be an efficient solution by Hofferek et al. in [129] to the synthesis problem presented in the previous section. In their approach, after expanding the Boolean variables, the negation of the specification formula was solved by a proof-producing SMT solver. After that, the synthesized control logic was extracted as Boolean relations [139] based on some form of Craig interpolation [77] from the produced refutation of the solver. However, it is not hard to see that with more universally quantified variables the transformation easily leads to an exponential growth of the formula size, and thus the applicability of this approach is limited.

Since the universally quantified variable is Boolean, one could consider it as a bit-vector with width of one. Reasoning about quantified bit-vectors in SMT solving is currently a subject of broad interest (see e.g. [140, 190, 234]). But again, Boolean variables have only two possible instantiations and the rest of the formula (including uninterpreted functions) does not really require bit-precise reasoning. Thus, that is not necessarily the best fitting logic to our problem.

6.1.3 Eager and Lazy Encodings

A very straightforward but relatively spacious solution method to formula $\mathcal{Q}.\mathcal{F}$ was not mentioned in the previous section. Equalities over uninterpreted functions is actually a background theory that has well-known eager translations into pure propositional logic. The transformation first eliminates every function application by introducing new Boolean variables for each of them and by adding explicitly additional constraints to guarantee their functional consistency [4,200]. Then the resulting formula is in equality logic, that can be directly translated into propositional logic and solved (following e.g. [62, 199] or [179]). Depending on whether the universally quantified variable is expanded, this eager encoding of $\mathcal{Q}.\mathcal{F}$ would result in an equisatisfiable SAT or quantified Boolean formula that can be solved directly by a SAT or QBF solver. In case the universally quantified variable is not eliminated, one has to pay attention to introduce the new abstraction variables into the right blocks of the quantifier prefix. The abstraction variables are all existentially quantified and each of them belongs to the quantifier block where the innermost variable on which they depend belongs to (see an example for that abstraction later in Section 6.1.4).

The previously described eager encoding to QBF was considered by Hofferek et al. in [129] and in [127] for the control logic synthesis problem. In their experiments the problems became infeasible due to the large amount of necessary additional transitivity constraints. Beyond the growth of formula size, this approach is practical only with background theories that have already well-known efficient encoding to propositional logic, thus it is not easily generalizable.

The dilemma of eager versus lazy encoding is very well known in the context of SAT and SMT solvers (see e.g. [114, 202]). One can choose to completely reduce a problem into propositional logic and then solve it with a single SAT call. The other option is to encode it only partially (e.g. construct a Boolean abstraction of the problem) and use a SAT solver as one of many collaborating decision procedures in a modular system [56, 150]. One justification for the eager encoding is that SAT solvers are efficient, fine-tuned tools that can handle millions of variables (of course, depending on the problem), while for some background theories the current theory solvers are not yet that efficient.

The same dilemma does not exist in the context of QBF solvers. In this context an eagerly encoded problem to solve is more difficult than SAT (assuming that NP $\neq$ PSPACE) [195,224]. Beyond that, both the underlying theory and the practical algorithms of QBF are not yet fully developed [165,167]. And thus the eager encoding of problems into QBF is yet rarely a practical option. But what about a lazy encoding? Is there a sound way to combine QBF reasoning with the theory solvers known from SMT systems? In the following sections we seek an answer for this question.

6.1.4 Lemmas on Demand for QBF modulo Theories

As a first step we illustrate how a "lazy" interplay between a QBF and a conjunctive theory solver would look like. After that, we will discuss some further possibilities to adapt this approach for less lazy solutions.

In the simplest – most lazy – setting we have a black-box certificate producing QBF solver coupled with an engine that decides the consistency of a set of theory literals. Though there are several QBF solving approaches with different underlying proof systems [91, 108, 143, 145, 149], for our demonstration we will use the extractable winning strategy [34, 103] found by the solvers, referring on them as (counter-)models of (un-)satisfiable formulas.

Let's take a look again on our toy example $\mathcal{Q}.\mathcal{F}$. The Boolean skeleton of the formula, noted as $\mathcal{P}.\mathcal{B}_\mathcal{F}$, is the following QBF:

$$\mathcal{P}.\mathcal{B}_\mathcal{F} := \exists\, b_{P(i)}, b_{P(f(i))}\ \forall\, c\ \exists\, b_{o=i}, b_{o=f(i)}, b_{P(o)}\ .\ b_{P(i)} \oplus b_{P(f(i))} \wedge (c \Rightarrow b_{o=i}) \wedge (\neg c \Rightarrow b_{o=f(i)}) \wedge b_{P(o)}$$

In this abstraction we ignore functional consistency (i.e. we did not use Ackermann's reduction [4]) and simply focus on the shape of the problem. For each predicate application and equality (i.e. theory atom), we introduce a new existentially quantified Boolean variable into the prefix of the abstraction (noted as $\mathcal{P}$). The quantifier block of the new variable depends on the arguments occurring in the encoded atom. For example atoms $P(i)$ and $P(f(i))$ refer only to the input variable, thus their abstraction belongs to the most outer quantifier block. The truth value of $o = i$ depends on c (because the value of o depends on c), thus $b_{o=i}$ must be part of the inner $\exists$-block. In CNF the matrix of the formula is $(b_{P(i)} \vee b_{P(f(i))}) \wedge (\neg b_{P(i)} \vee \neg b_{P(f(i))}) \wedge (\neg c \vee b_{o=i}) \wedge (c \vee b_{o=f(i)}) \wedge b_{P(o)}$.

Figure 6.1 depicts a possible tree-model found by a QBF solver to $\mathcal{P}.\mathcal{B}_\mathcal{F}$. Each path of that tree from root to leaf corresponds to a set of literals over EUF atoms (just like models found by SAT engines in an SMT solver). Invoking a theory solver on each of these sets can determine whether it is consistent w.r.t. the theory axioms (T-satisfiable) or not (T-unsatisfiable). In Figure 6.1 the background color of each set shows the result of this query (just like in Section 1.1.3). We can see that when c is assigned to be false, though the Boolean abstraction of $\mathcal{Q}.\mathcal{F}$ is satisfiable, the translation of the found assignment is T-unsatisfiable. The unsatisfiable core of that set of theory literals is $\{\neg P(f(i)), o = f(i), P(o)\}$ and thus the clause $(P(f(i)) \vee o \neq f(i) \vee \neg P(o))$ is a tautology. Since the found tree-model is not theory consistent, a new QBF solving iteration must start.

Learning the Boolean abstraction of that clause blocks the previously found solution (or at least one branch of it) in the QBF solver. Solving the refined QBF $\mathcal{P}.(\mathcal{B}_\mathcal{F} \wedge (b_{P(f(i))} \vee \neg b_{o=f(i)} \vee \neg b_{P(o)}))$ would again return true, with the tree-model presented in Figure 6.2. In this iteration the QBF solver flips the truth value of $b_{P(i)}$ and $b_{P(f(i))}$ to avoid the previous theory inconsistency. However,

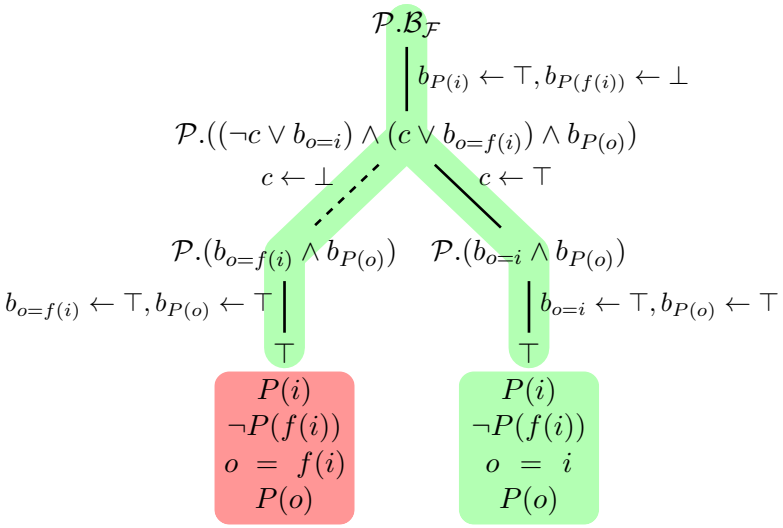


Figure 6.1: Tree model of formula $\mathcal{P}.\mathcal{B}_\mathcal{F}$. For each path of the tree the corresponding set of theory literals is constructed based on the truth value of the abstraction variables.

in this case the other branch, i.e. where c was assigned true, can not hold on the theory level. Thus, the negated Boolean abstraction of the new unsatisfiable core has to be learned and the extended QBF formula must be solved again.

After this extension, the new formula $\mathcal{P}.(\mathcal{B}_\mathcal{F} \wedge (b_{P(f(i))} \vee \neg b_{o=f(i)} \vee \neg b_{P(o)}) \wedge (b_{(P(i)} \vee \neg b_{o=i} \vee \neg b_{P(o)}))$ is proven false by the QBF solver. To check the correctness of the procedure (and to remind ourself that the original motivation is to synthesise c), one can extract a Herbrand function [18] $(c = P(f(i))\,?\,\top : \bot)$ from the found refutation. Replacing every occurrence of the universally quantified Boolean variable c in $\mathcal{Q}.\mathcal{F}$ with the atom $P(f(i))$ results in a T-unsatisfiable problem and so makes the negation of the formula valid. Going back to our original synthesis problem, if the negation of it is valid, it means that the reference design and the multipipelined design that uses the Herband function as control logic, are equivalent. Thus, with this lazy lemmas on demand [82] coupling of a QBF solver and a theory solver we refuted formula $\mathcal{Q}.\mathcal{F}$ and even synthesised a correct implementation for the control variable c.

6.1.5 Simple Refinements

One important benefit of the demonstrated procedure is that the internal behaviour of the QBF solver can be completely ignored and so an arbitrary certificate producing solver can be applied in the proposed workflow. Of course, giving up on this flexibility and willing to adapt the search engines allows a tighter collaboration that can reduce the amount of unnecessary work. In the context of SMT solvers there are several already established techniques (e.g. on-line SAT solving, support for propagation by theory solvers, incrementality) [194] that could improve our system as well.

For example, though in our demonstration always only one clause was learned,

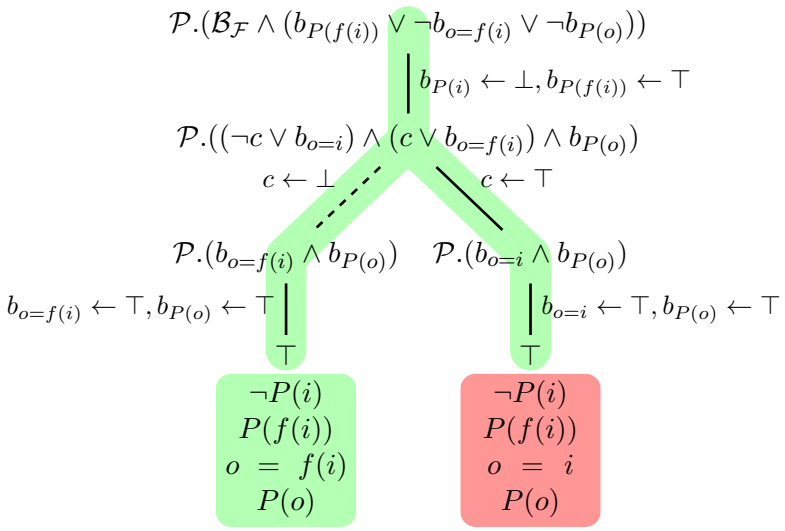


Figure 6.2: Tree-model of formula $\mathcal{P}.(\mathcal{B}_\mathcal{F} \wedge (b_{P(f(i))} \vee \neg b_{o=f(i)} \vee \neg b_{P(o)}))$ together with the corresponding theory literals for each branch of it.

it is not hard to see that in each iteration potentially more than one theory conflict can be found. Due to the tree structure of the found model, the branches can share many literals with each other and so an incremental theory solver can be beneficial to enumerate them.

Learning more clauses may reduce the number of necessary iterations, but does not change the fact that an increasing QBF formula is solved over and over again. Using an incremental QBF solver [163, 164], where new clauses can be added to the problem without starting the search from scratch, seems to be essential for an efficient lazy solver.

An even more efficient solution would be to invoke the theory check already during the QBF evaluation, i.e. move from a lazy lemmas on demand approach to a tighter integration. However, the implementation of this solution highly depends on the internals of the QBF solver. Expansion-based QBF solvers [13, 35, 48, 134, 135, 161] usually call a SAT solver at some point to evaluate the expanded formula under consideration. Replacing this SAT engine with an SMT solver opens up several interesting possibilities to gain a QBF modulo Theories solver. In case of search-based QBF solvers [65, 106, 162, 207, 238] there is no explicit SAT engine involved and thus another solution has to be looked for. In these solvers usually there is some form of cube learning (beyond the classical conflict driven clause learning), in order to prove true formulas [108, 157, 239]. Since our goal is to avoid theory inconsistent models, a possible way towards it is to guarantee that only theory consistent cubes are learned. In Section 6.1.7 we will show a possible implementation of that approach.

6.1.6 Incomplete Problem Solving

Although on our toy example a lemmas on demand approach worked, it does not mean that this is in general a solution method to our original problem. We had several implicit assumptions behind our approach inherited from classical SMT solution methods. For example, we assumed that it is sufficient to find a tree-model for the quantified Boolean abstraction of a problem such that every branch in that model is theory consistent in order to gain a model to the original quantified SMT problem. However, the following example demonstrates that this assumption actually does not hold. Consider the following simple formula

$$\mathcal{R}.\mathcal{G} := \exists\, t_1,\, t_2\ \forall\, c\ \exists\, t_3\ .\ (c \vee t_2 = t_3) \wedge (\neg c \vee t_2 \neq t_3) \wedge (t_1 = t_3)$$

where t_1, t_2 and t_3 are arbitrary terms, while c is a universally quantified Boolean variable. The Boolean skeleton of that formula, constructed as in the previous section, is the following QBF:

$$\mathcal{S}.\mathcal{B}_\mathcal{G} := \forall\, c\ \exists\, b_{t_2=t_3},\, b_{t_1=t_3}\ .\ (c \vee b_{t_2=t_3}) \wedge (\neg c \vee \neg b_{t_2=t_3}) \wedge (b_{t_1=t_3}).$$

A QBF solver would immediately return a tree-model like on Figure 6.3 for this formula. And so our QBF modulo Theories method would report that the formula is true. However, taking a closer look on $\mathcal{R}.\mathcal{G}$ shows that, considering the

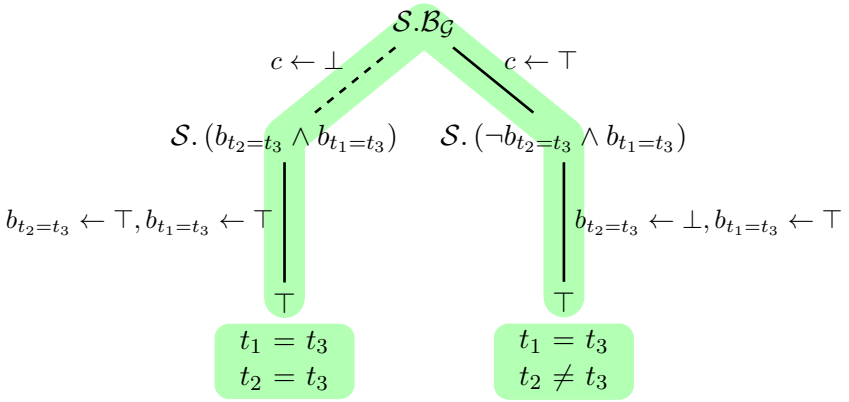


Figure 6.3: Tree-model constructed for formula $\mathcal{S}.\mathcal{B}_\mathcal{G}$ where both branches are found theory consistent.

axioms of equivalence, this formula is actually false. Expanding the universally quantified variable leads to the following simple problem in equality logic:

$$\begin{aligned}\mathcal{R}.\mathcal{G} &\equiv \exists\, t_1\, t_2.\ (\exists t_3.\ t_2 \neq t_3 \wedge t_1 = t_3) \wedge (\exists t_3'.\ t_2 = t_3' \wedge t_1 = t_3') \\ &\equiv_{sat}\ t_2 = t_3' \wedge t_3' = t_1 \wedge t_1 = t_3 \wedge t_2 \neq t_3.\end{aligned}$$

The conjunction of the two possible instantiations of c actually builds an implicit transitivity chain between variables t_2 and t_3 (via t_1) and so contradicts $t_2 \neq t_3$.

This example proved that our problem abstraction, even after complete refinement by our method, does not imply our original problem. Thus, our method is at least incomplete. And from that follows that when our solver proves a formula true, the more proper answer regarding the truth of the original problem would be "unknown".

But what about when our simplistic solving method returns unsat? Are these answers correct? To show that the presented procedure does not claim unsatisfiability of true formulas, we need to take a closer look on that scenario. In case the original problem is true, we can always consider the equivalent formula where every universally quantified Boolean variable is explicitly instantiated and all the inner variables are accordingly renamed. From any satisfying assignment of that formulation, a set of ordered truth assignments can be extracted such that they together define a tree-model to our abstraction (after mapping back the renamed variables to branches of that tree). And so, assuming that a correct QBF solver is used, the existence of this winning strategy implies that the solver must answer true to the first Boolean abstraction. Since every learned clause is valid w.r.t. the theory axioms, and the original formula has a theory consistent solution, these clauses might change the model but not the satisfiability of the abstract QBF. Thus in this scenario our lemmas on demand method can not derive false.

6.1.7 `QBeq`: A Prototype QBF modulo Theories Solver

To gain a proof of concept and to learn the potentials and limitations of an incomplete QBF modulo Theories solver, we developed a prototype engine called `QBeq` that can take as an input `Q`uantified `B`oolean formulas with `EQ`ualities over uninterpreted functions. The solver combines through their APIs the search-based QBF solver `depQBF` [162] (version 5.0) with `MathSAT5` [73] (version 5.6) that is responsible for the theory reasoning.

The main idea behind the implementation is to intercept the QBF solver every time when it finds a satisfying truth assignment (also called a cover set) for the matrix of the abstraction formula. Since these cover sets are the base of cube learning in the solver, guaranteeing their theory consistency implies that every learned and derived cube will be theory consistent as well. And so, whenever the QBF solver finds a solution, we invoke a theory solver to check the theory consistency of it. In case the solution is T-satisfiable, the QBF solver can proceed as usual (e.g. learn a $\exists$-reduced cube from it). When the found cover set is T-unsatisfiable, the QBF solver is not allowed to consider it as a solution. Thus, in that case instead of learning a cube, it must learn a theory lemma (i.e. the negation of the abstraction of the T-unsatisfiable core) as a new clause. Since this new clause will be falsified by the current assignment, the QBF solver is forced to search for another solution. Florian Lonsing, the developer of `depQBF`, proposed and developed for us a callback function based solution in `depQBF 5.0` to achieve our desired timely interception and cover set

inspection. Thanks to his modifications in `depQBF`, implementing a tighter QBF modulo Theories solver became significantly easier.

Our prototype solver exploits these modifications of `depQBF` by implementing and registering the functions that are meant to be called when `depQBF` finds a cover set. `QBeq` first parses the input SMT2 [26] file containing a quantified Boolean formula with equalities. As a next step, it constructs the QBF skeleton of the problem and initializes the formula in `depQBF` based on it. At the same time, in the SMT solver an empty formula is initialized where the signature of every term of the problem is declared. The remaining part of the solver simply invokes the SMT solver with a set of theory literals (in form of a conjunct of assumptions) whenever the QBF solver finds a cover set. In case the SMT solver finds it satisfiable, `depQBF` is informed and continues as nothing would have happened. If the set of theory literals were unsatisfiable, the Boolean abstractions of the literals in the unsatisfiable core are collected by `QBeq` and forwarded to `depQBF` as a theory lemma. Since this lemma invalidates the current solution of `depQBF`, it backtracks and continues the search for another assignment or returns in case a refutation of the formula was found.

6.1.8 Preliminary Experiments

The main practical motivation behind our work is the control logic synthesis problem described in Section 6.1.1. We believe that our proposed QBF modulo Theories approach is a step towards it, but several milestones to address these problems (e.g. certificates or preprocessing) are still missing. Thus, first we pick a small and manageable part of that problem to focus on. To evaluate our prototype solver and to estimate the performance of alternative solution methods, we generated a set of similar but much simpler problems. Since our targets are problems where potentially many universally quantified Boolean variables are combined with relatively simple theory atoms, we started our problem generation from already existing QBF formulas. We considered two families of QBF problem instances from the formula library called QBFLIB [107] of the annual QBF evaluations [204] as starting points, both containing problems only with ∃-∀-∃ quantifier prefix structure. The benchmark family `ToiletC` contains 85 problems from the domain of planning [68], while the 303 formulas of the `Abduction` family are originating from SAT problem instances. All the experiments were performed with time limit of 300 seconds and memory limit of 7 GB on our cluster with Intel Xeon E5-2620 v4 @ 2.10GHz CPUs.

As a first step we measured the performance of `depQBF` on these pure QBF problems, to get a base line to our further experiments. Since `QBeq` uses `depQBF 5.0` with a specific set of options (the list of options is `--dep-man=simple --incremental-use --no-qbce-dynamic --traditional-qcdcl` and `--no-pure-literals --no-qpup-sdcl`) we compared this version to the most recent version (`depQBF 6.03`) under default settings. Table 6.1 summarizes the results. As we can see, the `ToiletC` problems are mostly really easy to solve

Table 6.1: Results of different versions of `depQBF` on the QBF problem families `Abduction` and `ToiletC`.

Solver	`Abduction` (303)				`ToiletC` (85)			
	Solved	SAT	UNSAT	Time	Solved	SAT	UNSAT	Time
`depQBF 5.0`	287	152	135	1081.1	82	29	53	202.6
`depQBF 6.03`	288	153	135	4961.9	82	29	53	202.0

to both versions and the proportion of SAT and UNSAT instances is roughly 1:2. The `Abduction` problems are more time consuming to solve, but all in all the numbers of solved instances by the two versions are similar (though `depQBF 6.03` required more time to solve them).

As a next step we simply transformed each of the QBF problems into the input format of SMT solvers (see the specification of the SMT2 language in [26]), without any change to the formulas. Each of the resulting SMT problems consists of a single assertion with an explicit quantifier prefix containing only Boolean variables and the clauses of the source QBF matrix. These formulas (denoted as `Abduction-SMT` and `ToiletC-SMT`) are then heavily quantified SMT problems, but without any underlying background theory to consider. We tried different state-of-the-art solvers to evaluate the alternative solution methods described in Section 6.1.2. Our main goal here was not to show the good or bad performance of the solvers, but rather to draw attention on how different they can be. As theorem prover we used `Vampire` [148] (version 4.4) and as SMT solver we experimented with `CVC4` [25] (version 1.7) and `z3` [81] (version 4.8.7), because they both support quantified formulas. In case of `Vampire` we used the portfolio solver setting (`--input_syntax smtlib2 --mode portfolio -p off -t 300 --schedule casc_2019`), while `z3` was invoked without any special options. In case of `CVC4` we tried different options based on their SMT competition configurations. In this first experiment the effect of the used options of `CVC4` was minor, nevertheless the best results (that are presented here) were achieved with the options `--decision=internal --simplification=none --no-inst-no-entail --no-quant-cf --full-saturate-quant`. This experiment already includes our tool `QBeq` as well. Table 6.2 shows the number of solved instances for each solver. In the case of the problems in `ToiletC-SMT` there is no significant differ-

Table 6.2: Results of different solvers on the QBF problems `Abduction` and `ToiletC` reformulated as SMT problems.

Solver	`Abduction-SMT` (303)				`ToiletC-SMT` (85)			
	Solved	SAT	UNSAT	Time	Solved	SAT	UNSAT	Time
`z3 4.8.7`	1	1	0	0.7	83	29	54	256.3
`CVC4 1.7`	239	112	127	8300.5	80	28	52	361.6
`Vampire 4.4`	0	0	0	0.0	81	25	56	661.5
`QBeq`	284	155	129	2239.9	82	29	53	213.9

ence between the solvers. For the formulas of `Abduction-SMT` the solvers react differently. On these problems both `z3` and `Vampire` chose a wrong strategy and run out of time. On the other hand, here `CVC4` performs really well compared to

the other solvers. The last row in Table 6.2 shows the number of solved instances by our tool `QBeq`. Since the problems do not have background theories (and so there are no theory induced conflicts), the Boolean abstraction of each problem is the exact same QBF problem that we started with. And so the performance of the tool is expected to be close to `depQBF 5.0` (in Table 6.1). In the case of `ToiletC-SMT` instances the increase in running time is negligible, while in case of `Abduction-SMT` an overhead produced by the trivial SMT queries is observable.

The comparison in Table 6.2 is rather unfair since we try to solve QBF problems with SMT solvers and theorem provers. So as a next step we introduced theory atoms into our problems to get closer to the meant domain of the solvers. We generated from each QBF problem a brand new random SMT problem containing universally quantified Boolean variables and equalities between existentially quantified variables. In each QBF problem we replaced in the formula prefix every existentially quantified Boolean variable with two new existentially quantified (uninterpreted) variables. The universally quantified Boolean variables neither in the prefix nor in the formula matrix were changed. In the formula matrix we replaced every existentially quantified Boolean variable with a random equality between two different uninterpreted variables. One of these two variables is always from the quantifier block of the replaced Boolean variable, while the other variable is either from the same or from the outer block (in case the replaced Boolean variable was in the inner block).

The intention was to gain quantified SMT formulas with similar QBF abstractions as our original QBF formulas were. As a consequence, many of the resulting problems are likely to be refutable already on the Boolean layer (since was constructed based on a false QBF formula) without considering any theory reasoning. This benefits our tool in a comparison, thus the more interesting to see here is how the other solvers attempt to solve these problems.

Table 6.3 presents the found results. First of all, we observed that `Vampire` in that experiment produced spurious results (e.g. refuted each problem in `ToiletC`, though 20 of them is true). Since the problems are formulated in the language of SMT solvers, which is not the primal input format of `Vampire`, we suspect that the parsing is not perfect yet. Thus, here we do not present the results produced by this theorem prover. Considering the SMT solvers, `CVC4`

Table 6.3: Results of different solvers on the quantified equality problems generated randomly from the QBF benchmark families `Abduction` and `ToiletC`.

Solver	`Abduction-EQ` (303)				`ToiletC-EQ` (85)			
	Solved	SAT	UNSAT	Time	Solved	SAT	UNSAT	Time
`z3 4.8.7`	22	0	22	1701.1	85	20	65	69.1
`CVC4 1.7`	33	0	33	137.0	66	1	65	367.5
`QBeq`	283	125	158	2065.6	82	29	53	264.7

handled the `Abduction-EQ` problems better than `z3`, by solving 33 unsatisfiable formulas from the 303. On the other hand, `z3` performed exceptionally well on the `ToiletC-EQ` problem instances. It solved every formula and showed that 20

instances are true, while 65 are refutable. On these problems the setting used by CVC4 was not as successful (solved 13 unsatisfiable and 1 satisfiable instances from the 85). In Table 6.3 we report for the ToiletC-EQ problems the results of CVC4 with the options --pre-skolem-quant --full-saturate-quant, that produced better results. Our tool needed more time (but similar to the pure QBF instances) to address these problems, and actually 3 instances remained unsolved by it (just like in the QBF case). What is more interesting is that QBeq found 29 SAT instances, and based on the results of z3, we know that 9 of them must be false but the weak abstraction was not sufficient to show it. Thus, considering only the ToiletC-EQ problems, z3 seems to provide a quick and exact solution method. However, in case of the Abduction-EQ problem set the SMT solvers were not really successful, while QBeq successfully identified 158 false formulas and could not solve only 20 instances. At the 125 instances where QBeq returned true, we do not know what is actually the correct answer. Nevertheless, all in all we can draw the conclusion that QBeq can be an efficient asset in the case of the unsatisfiable problems.

In the last experiment we transformed further our generated quantified equality problem instances. Replacing every sort in the problems with bit-vectors yields a set of quantified bit-vector problems (containing only equality operations over bit-vectors). For each problem we estimated an upper bound on the number of necessary bits from the number of existentially quantified variables. Then, we generated a new problem for each possible bit-width (the universally quantified Boolean variables always remained 1 wide). With lower widths the problems are more likely unsatisfiable, but hopefully easier to solve [141]. The result was 3127 Abduction based problems (noted as Abduction-EQ-BV) and 688 ToiletC based problems (called ToiletC-EQ-BV). Since QBeq and Vampire do not support bit-vectors, they did not participate in that experiment. The performance of the SMT solvers on these problems are presented in Table 6.4. Since here the problems contain quantified bit-vectors, the used setting of CVC4

Table 6.4: Results of SMT solvers on the quantified equality problems generated randomly from the QBF benchmark families Abduction and ToiletC, encoded as quantified bit-vector SMT problems.

Solver	Abduction-EQ-BV (3127)				ToiletC-EQ-BV (688)			
	Solved	SAT	UNSAT	Time	Solved	SAT	UNSAT	Time
z3 4.8.7	199	0	199	15674.2	688	159	529	7235.1
CVC4 1.7	228	9	219	7915.3	247	46	201	4767.1

was changed accordingly. The presented results were produced with the options --full-saturate-quant --no-cbqi-innermost. Considering first the problems in Abduction-EQ-BV, CVC4 identified 9 different instances as satisfiable (each of them with width 1), while refuted 219 formulas (over 60 different problem instances). The solver z3 solved a slightly less 199 formulas over 42 different instances (approximately in twice as much time as CVC4) and found all unsatisfiable. Regarding the problems of ToiletC-EQ-BV, CVC4 solved 247 from

them, finding 46 satisfiable formulas (over 7 different instances) and 201 unsatisfiable formulas (over 42 different instances). On the other hand, `z3` was again really fast with the `ToiletC` based problems and solved all of them. All the 159 satisfiable instances belong to the 20 satisfiable problems found in Table 6.3.

So while the problems of `ToiletC-EQ` could be tackled efficiently with SMT solvers (either as is or as quantified bit-vectors), in case of the formulas of `Abduction-EQ` the current off-the-shelf solution approaches were not sufficient. All in all we can draw the conclusion that our logical fragment of interest can be addressed in several different ways, but in many cases they represent a challenging subset of problems. And because of that, it is reasonable to look for alternative solutions (like our proposed QBF modulo Theories approach) even if they are only refutationally complete, as long as they are sufficiently efficient.

6.1.9 Future Work and Open Questions

In this section we presented an idea to combine theory reasoning with QBF solving, motivated by a synthesis problem. Though the idea is simple, it raises several interesting theoretical and practical questions that must be considered. The proper way to do it would be to develop an abstract framework where one can formally reason about QBF solving. This framework would allow us to analyse and understand the currently implicit details. Starting from there, just as it was done in the case of SMT technology, one could extend the framework to abstract QBF modulo Theories and investigate correctness and termination criteria. Unfortunately, the diversity of orthogonal QBF solution methods and the yet rather immature level of experience with these systems make this first step very challenging. To simplify this step, our focus was first only on search-based QBF solvers. Even for these systems a realistic abstract formalization is rather challenging to find. An obvious future work is to continue this abstraction attempt and even extend it to alternative solving principles.

Focusing on the problems rather than on the solvers, an interesting question is how hard are our formulas really. Although here we considered only the simplest possible background theory, the goal would be to support more general, theory-rich problems. Nevertheless, the proper definition of the logic fragment and the complexity analysis of it remains future work.

A rather practical aspect of the problem that was not addressed here is the question of pre- and inprocessing. Formula simplifications are a mandatory element of most solving tool chains and thus we have to investigate which of these steps are allowed in the presence of theory atoms. But for that, again, we need to understand the solving principles and the problems.

Bearing in mind that at the end of the day we would like to synthesise components with our system, the question of certificate production is inescapable. Future work must take a closer look on what are the possibilities regarding that.

During the illustration of our proposed solution method we saw that the current way of problem abstraction is too weak (see Section 6.1.6). It is a natural

question whether there are better ways to construct the Boolean skeleton of the problem, such that we can capture more its real semantics. For example, one could introduce more than one Boolean variable into different prefix blocks to describe the same relational atom. Forcing them to be equivalent in the Boolean layer, the produced models may trigger more theory conflicts. Investigating further the difference between lazy and eager encodings in our context may reveal some new approaches.

If we consider our approach to capture only some of the unsatisfiable problems, the question is in what tool chain it could be beneficial. Considering our method as a first step in a layered or CEGAR approach seems to be reasonable. Complementing our method with explicit bit-blasting with smaller widths could be an efficient way to identify the easily provable and refutable problems and so invoke more expensive, complete methods only on the harder problems.

An important task for future work is to find further application domains where a QBF modulo Theories solver could be employed. Our hope is that as QBF solvers will become faster and faster, our line of research will become more relevant and applicable as well.

6.2 Maximum Satisfiability and Theory Reasoning

In Chapter 3, as a component of our combined MaxSMT solver, we introduced an abstract framework to describe the internal behaviour of such SAT (and SMT) solvers that can evaluate formulas under assumptions. An important feature of these solvers is that for unsatisfiabile formulas they can return a subset of the assumptions that were needed for the refutation. In our paper we claimed soundness and termination of the proposed framework, but omitted proofs for these claims, due to limitations in space. This section will focus on these proofs.

6.2.1 Correctness of A-Sat

Our framework extends assumption based SAT solvers with theory reasoning similar to how SAT solvers are combined with theory solvers in [194]. Consequently, the reasoning about the correctness of the framework is also similar. The main difference is that the set of decision literals is extended with the current set of assumptions in every state of the derivation. Further, we introduced two new rules to capture explicitly the states when the derivation terminates (i.e. when either a solution or a refutation is found). Although these differences are rather technical, for the sake of completeness we present the necessary lemmas and theorems to show the soundness of this calculus.

Lemma 6.2.1. *If* $A \mid \emptyset \mid F \overset{\textbf{A-Sat}}{\Longrightarrow}{}^{*} A \mid M \mid G$*, then the following hold.*

P-1. *All atoms in* A*, in* M *and in* G *are atoms of* F*.*

P-2. AM *contains no literal more than once and is indeed an assignment, that is, it contains no pair of literals of the form* ℓ *and* $\neg\ell$*.*

P-3. G *is logically equivalent to* F*.*

P-4. *If* M *is of the form* $M_0 l_1 M_1 \dots l_n M_n$*, where* $l_1, \dots, l_n$ *are all the decision literals of* M*, then* $F, A, l_1, \dots, l_i \models M_i$ *for all* i *in* $0 \dots n$*.*

Proof. In an initial state $A \mid \emptyset \mid F$ all properties hold, since A is an assignment over $atoms(F)$. To show that all **A-Sat** rules preserve these properties, consider $A \mid M' \mid F' \overset{\textbf{A-Sat}}{\Longrightarrow} A \mid M'' \mid F''$ as a step, where all properties hold in $A \mid M' \mid F'$.

The first two properties trivially hold, because A can not be changed during any derivation and any new atoms of M'' or F'' are the ones in F' or M', all of which belong to F. Third property could be violated only by the Learn or Forget rules, but they can only add or remove logical consequences of F' preserving the logical equivalence of F' and F''.

For the fourth property, let M' be in the form of $M'_0 l^d_1 M'_1 \dots l^d_n M'_n$, where $l^d_1, \dots, l^d_n$ are all the decision literals. Rules Decide, Learn and Forget trivially maintain the property, since in case of Decide there is nothing to prove, while Learn and Forget do not change AM' and ensure that F' and F'' are

logically equivalent. If the applied rule is UnitProp, we have that F' is unchanged so $F'' = F'$ while M'' is $M'\ell = M'_0 l_1^d M'_1 \dots l_n^d M'_n \ell$, where F', and so F'', contains a clause $(C \vee \ell)$ and $AM'_0 l_1^d M'_1 \dots l_n^d M'_n \models \neg C$. By our inductive assumption, $F'', A, l_1^d, \dots, l_i^d \models M'_i$ for $i \leq n$ since $F'' = F'$. M'_n has been extended to become $M'_n \ell$, but $F'', A, l_1^d, \dots, l_n^d \models M'_1 \dots M'_n$, $F'' \models (C \vee \ell)$ and $A, M'_0 l_1^d, M'_1, \dots, l_n^d, M'_n \models \neg C$ implies that $F'', A, l_1^d, \dots, l_n^d \models M'_n, \ell$. Hence, UnitProp preserves property 4.

Finally, if the applied rule is Backjump, AM' must have the form $AM'_0 l_1^d M'_1 \dots l_{i-1}^d M'_{i-1} l_i^d N$ and there is a clause $C \in F'$ that is falsified by AM', i.e. $AM'_0 l_1^d M'_1 \dots l_{i-1}^d M'_{i-1} l_i^d N \models \neg C$. From the side condition of the rule we know that there is a clause $(C' \vee \ell')$ s.t. $F' \models C' \vee \ell'$ and $AM'_0 l_1^d M'_1 \dots l_{i-1}^d M'_{i-1} \models \neg C'$. M'' is of the form $M'_0 l_1^d M'_1 \dots l_{i-1}^d M'_{i-1} \ell'$ where $l_1^d, \dots, l_{i-1}^d$ are all the decision literals of M''. As in UnitProp $F'' = F'$ and thereby $F'', A, l_1^d, \dots l_j^d \models M'_j$ for $j \leq i-1$ since $F'' = F'$. M'_{i-1} has been extended to become $M_{i-1}\ell'$, but since $F'', A, l_1^d, \dots l_{i-1}^d \models M'_1 \dots M'_{i-1}$, $F'' \models (C' \vee \ell')$, and $AM'_0 l_1^d M'_1 \dots l_{i-1}^d M'_{i-1} \models \neg C'$, we also have $F'', A, l_1^d, \dots l_j^d \models M'_j \ell'$. Hence, Backjump preserves P-4. □

Lemma 6.2.2. *Assume that* $A \mid \emptyset \mid F \overset{\textbf{A-Sat}}{\Longrightarrow}{}^* A \mid M \mid F'$ *and that there is a clause* D *in* F' *s.t.* $AM \models \neg D$. *Then either* Unsat *or* Backjump *applies to* $A \mid M \mid F'$.

Proof. If M contains no decision literals, then we have that $F, A \models M$ and thus $F, A \models \neg D$ by P-4. of Lemma 6.2.1. Since F' is equivalent to F (due to P-3. in Lemma 6.2.1) and $D \in F'$ we can conclude that $F \wedge A$ is unsatisfiable and so $F \models \neg A$. So there must exist some clause $C \subseteq \neg A$ s.t. $F \models C$ and $A \models \neg C$ (e.g., we can resolve away the forced literals in D until we obtain only literals of $\neg A$). Hence, in this case Unsat applies. If M contains the decision literals $l_1^d, \dots, l_n^d$ we can construct a backjumping clause $C' \vee \ell'$ by using standard 1-UIP clause learning techniques starting with the conflict clause D. P-4. in Lemma 6.2.1 shows that AM has only decision literals and literals forced by prior decisions. Hence, the 1-UIP learning technique can be applied by resolving away all but one forced literal, ℓ', at the deepest level. This necessarily yields a new clause $(C' \vee \ell')$ satisfying the conditions of Backjump, and thus Backjump applies. □

Lemma 6.2.3. *If* $A \mid \emptyset \mid F \overset{\textbf{A-Sat}}{\Longrightarrow}{}^* S$ *and* S *is final with respect to* ***A-Sat****, then* S *is either* $\mathit{conflict}(F', C)$ *or* $\mathit{SAT}(AM, F')$.

Proof. Assume S is a final state with respect to **A-Sat**, but it is neither $\mathit{conflict}(F', C)$ nor $\mathit{SAT}(AM, F')$. Then S has to have the form $A \mid M \mid F'$. We know that there is no clause C in F' s.t. $AM \models \neg C$, otherwise either Unsat or Backjump would be applicable (Lemma 6.2.2). We also know that all literals of F' are defined by AM, otherwise Decide would be applicable. Thus, AM has to be a model of F'. But then Sat-Model is applicable, which contradicts the assumption that S is final. □

Theorem 6.2.4 (Soundness of **A-Sat**). *For any derivation $A \mid \emptyset \mid F \Longrightarrow \cdots \Longrightarrow S$ in **A-Sat**, where S is final with respect to **A-Sat**, we have that*

1. *$S = \mathit{conflict}(F', C)$ with $F' \models C$, $A \models \neg C$ iff $F \wedge A$ is unsatisfiable.*
2. *If $S = \mathit{SAT}(AM, F')$ then AM is a model of F.*

Proof. From Lemma 6.2.3, we know that there are two possible final states of a derivation in **A-Sat**. First consider the case when S is $\mathit{conflict}(F', C)$. Then due to the side condition of Unsat, $AM \models \neg D$ for a clause $D \in F'$ and M contains no decision literals. Further, $F \models D$ (P-3. of Lemma 6.2.1) and $F, A \models \neg D$ (P-4. of Lemma 6.2.1), i.e. $F \wedge A$ is unsatisfiable. For the other direction, if $F \wedge A$ is unsatisfiable, we have that $A \models \neg F$ and so there is no possible extension of A (i.e. possible M) s.t. $AM \models F$ holds.

If S is the state $\mathit{SAT}(AM, F')$, the condition of rule Sat-Model and P-3. in Lemma 6.2.1 lead to the conclusion that AM is a model of F. □

Theorem 6.2.5 (Termination of **A-Sat**). *Any sequence of transitions in **A-Sat** starting from state $A \mid \emptyset \mid F$ that contains no infinite subsequence consisting only of rules from the set $\{\text{Learn}, \text{Forget}\}$ is finite.*

Proof. See proof of Theorem 2.10 in [194]. Extending the introduced strict partial ordering of the states to our new additional states ($\text{SAT}(AM, F')$ and $\text{conflict}(F', C)$) by considering them as minimal elements closes the proof. □

6.2.2 Correctness of A-Smt

Notice that our proposed **A-Smt** calculus is minimalist in a sense that it does not contain a rule to restart or to propagate via theory lemmas. Thus, after establishing that our **A-Sat** calculus is correct, the soundness and termination of **A-Smt** is relatively straightforward to prove.

Lemma 6.2.6. *If $A \mid \emptyset \mid F \overset{\textbf{A-Smt}}{\Longrightarrow}{}^{*} A \mid M \mid G$, then the following hold.*

P-1. All atoms in A, in M and in G are atoms of F.

P-2. AM contains no literal more than once and is indeed an assignment, that is, it contains no pair of literals of the form ℓ and $\neg\ell$.

P-3. G is T-equivalent to F.

P-4. If M is of the form $M_0 l_1 M_1 \ldots l_n M_n$, where $l_1, \ldots, l_n$ are all the decision literals of M, then $F, A, l_1, \ldots, l_i \models_T M_i$ for all i in $0 \ldots n$.

Proof. As for Lemma 6.2.1 for the **A-Sat** calculus. □

Lemma 6.2.7. *If $A \mid \emptyset \mid F \overset{\textbf{A-Smt}}{\Longrightarrow}{}^{*} A \mid M \mid F'$ and AM is T-inconsistent, then either there is a conflicting clause in F' or else T-Learn applies to $A \mid M \mid F'$, generating a conflicting clause.*

Proof. If AM is T-inconsistent, then there exists a subset $\{l_1, \ldots, l_n\}$ of AM s.t. $\emptyset \models_T \neg l_1 \vee \cdots \vee \neg l_n$. This clause is either already in F' or can be learned by one T-Learn step. Note, that in case $\{l_1, \ldots, l_n\} \subseteq A$ and M still has decision literals, our current calculus does not allow to apply Unsat, but only T-Backjump. However, in that case learning $\neg l_1 \vee \ldots \neg l_n$ ensures that any decision will lead to a conflict, and so it is guaranteed that a state without decision literals in M will be reached. □

Lemma 6.2.8. *Assume that* $A \mid \emptyset \mid F \overset{\textbf{A-Smt}}{\Longrightarrow}{}^{*} A \mid M \mid F'$ *and* $AM \models \neg D$ *for some clause* D *in* F'*. Then after finitely many* T-Learn *steps, either* Unsat *or* T-Backjump *applies to* $A \mid M \mid F'$*.*

Proof. In case there are still decision literals in M, the proof is just as in Lemma 6.2.2 and so without an explicit T-Learn step, rule T-Backjump is applicable. If M contains no decision literals, we have that $F, A \models_T \neg D$ due to P-4. of Lemma 6.2.6. Since F' is T-equivalent to F (P-3. in Lemma 6.2.6) and $D \in F'$ we can conclude that $F \wedge A$ is T-unsatisfiable and so $F \models_T \neg A$. Then some subset of $\neg A$ is either already in F' (and so Unsat is directly applicable as in Lemma 6.2.2), or can be learned by one T-Learn step resulting in a state where Unsat is applicable. □

Theorem 6.2.9 (Soundness of **A-Smt**)**.** *For any derivation* $A \mid \emptyset \mid F \Longrightarrow \cdots \Longrightarrow S$ *in* ***A-Smt*** *where* S *is final with respect to* ***A-Smt*** *we have that*

1. $S = \mathit{conflict}(F', C)$ *with* $F' \models C$, $A \models \neg C$ *iff* $F \wedge A$ *is T-unsatisfiable.*
2. *If* $S = \mathit{T\text{-}SAT}(AM, F')$ *then* AM *is a T-model of* F*.*

Proof. As for Theorem 6.2.4 for the **A-Sat** calculus, building on the above introduced **A-Smt** lemmas. □

Theorem 6.2.10 (Termination of **A-Smt**)**.** *Any sequence of transitions* $A \mid \emptyset \mid F \Longrightarrow \cdots$ *in* ***A-Smt*** *that contains no infinite subsequence consisting only of rules from the set* {T-Learn, T-Forget} *is finite.*

Proof. As in Theorem 6.2.5 for **A-Sat**. □

6.3 Incremental SAT Solving and Inprocessing

In this section we discuss further details and present some theoretical extensions of the formal framework that was published in [94] and was presented in Chapter 4. However, regarding the formalization of the reconstruction function there is an important difference in the current presentation compared to [94]. In the upcoming sections we will consider and use the following non-recursive definition of reconstruction function, which was proposed by Christoph Scholl during working on [94].

Definition 6.3.1 (Reconstruction Function)**.** *Given a truth assignment τ and a witness labelled clause $(\omega : C)$ the reconstruction function is defined as*

$$\mathcal{R}_{(\omega:C)}(\tau) = \begin{cases} \tau & \text{if } \tau(C) = \top \\ \tau \circ \omega & \text{otherwise.} \end{cases}$$

The reconstruction function for a sequence of witness labelled clauses σ is defined as $\mathcal{R}_\varepsilon = id$ and $\mathcal{R}_\sigma = \mathcal{R}_{(\omega_1:C_1)\cdot\ldots\cdot(\omega_n:C_n)} = \mathcal{R}_{(\omega_1:C_1)} \circ \ldots \circ \mathcal{R}_{(\omega_n:C_n)}$.

One of the benefits of that simplified formalization is that it directly builds on function composition, which is an associative operator, thus for a given truth assignment τ and a sequence of witness labelled clauses $\sigma{\cdot}\sigma'$ we have immediately that $\mathcal{R}_{\sigma\cdot\sigma'}(\tau) = \mathcal{R}_\sigma(\mathcal{R}_{\sigma'}(\tau))$, while in [94] we needed Lemma 4.4.3 to show that property. The new formalization does not change the meaning or the properties of the function, but highly simplifies the presentation, thus we will use it here.

6.3.1 Refined Reconstruction Property

Another point where our formalization could be refined is regarding the reconstructive property defined in Definition 4.2.4 in [94]. In our paper this property was defined such that every satisfying truth assignment of the irredundant clauses must after solution reconstruction satisfy all the irredundant clauses conjoined with the clauses on the reconstruction stack. This is a convenient definition, since most of our theorems are considering the combined formula $\varphi \wedge \sigma$ during inprocessing. But in practice what we actually care about is the satisfaction of our original input formula, independently from the current state of the derivation. In our incremental inprocessing calculus the input formula F^i of each iteration i is logically equivalent with $\varphi^i \wedge \sigma^i$ in every step of the derivation (see Cor. 4.4.1 in Chapter 4). However, this property might change if we for example extend the framework with non-model-preserving clause learning. Therefore here we introduce a refined definition of reconstruction property that is easier to use in a context where this formula equivalence is not maintained.

Definition 6.3.2 (Reconstruction Property)**.** *An abstract state $\varphi\,[\,\rho\,]\,\sigma$ satisfies the reconstruction property w.r.t. a formula F iff for all truth assignments τ satisfying φ, the result of the reconstruction function $\mathcal{R}_\sigma(\tau)$ is a satisfying assignment of F.*

Now we show that this refined reconstruction property is also maintained in our incremental inprocessing calculus. The proof is very similar to the one presented in [94], but requires less helping propositions or lemmas along the way, especially when we reason about strengthening a formula.

Theorem 6.3.1 (Incremental Reconstructiveness)**.** *In any derivation starting from the initial state in the incremental calculus of Fig. 4.5, the reconstruction property w.r.t. F^i holds in each phase $i = 0 \ldots n$ and in that for each state j with $0 \leq j \leq k_i$.*

Proof. In the initial state $i, j = 0$, $\varphi_0^0 = F^0$, $\sigma_0^0 = \varepsilon$, and so for any satisfying assignment τ of F^0, $\mathcal{R}_\varepsilon(\tau)(F^0) = \top$. Assume that in a state j of a phase i (where $0 \leq j < k_i$ and $0 \leq i \leq n$), the reconstruction property (i.e. $\forall\tau.(\tau(\varphi_j^i) = \top \Rightarrow \mathcal{R}_{\sigma_j^i}(\tau)(F^i) = \top)$) holds. Notice that F^i is not changed by any of the rules except AddClauses (when we increase i and start a new phase as well). If the applied rule is Learn$^-$ or Forget, neither φ_j^i, nor σ_j^i changes and so the property is trivially maintained. Rule Drop removes an entailed clause of φ_j^i, while σ_j^i is unchanged and so the property remains. Same holds when rule Strengthen is applied, since in that case any satisfying assignment of $\varphi_{j+1}^i = \varphi_j^i \wedge C$ satisfies φ_j^i as well, while the reconstruction stack remains the same as before. In case of rule Weaken$^+$, consider an arbitrary truth assignment τ' such that it satisfies $\varphi_{j+1}^i = \varphi_j^i \setminus C$. If there is no such assignment, the reconstruction property trivially holds. Otherwise, we need to consider two possibilities. If $\tau'(C) = \top$, then $\tau'(\varphi_j^i) = \top$, thus $\mathcal{R}_{\sigma_j^i}(\tau')(F^i) = \top$ by induction. Further, due to the definition of the reconstruction function (see Def. 6.3.1), $\mathcal{R}_{\sigma_{j+1}^i}(\tau') = \mathcal{R}_{\sigma_j^i \cdot (\omega:C)}(\tau') = \mathcal{R}_{\sigma_j^i}(\tau')$ because $\tau'(C) = \top$ and thus $\mathcal{R}_{\sigma_{j+1}^i}(\tau')(F^i) = \top$. The other possible case is that $\tau'(C) \neq \top$. In that case from the side condition of Weaken$^+$ and from Prop. 4.2.1 together it follows that $(\tau' \circ \omega)(\varphi_j^i) = \top$ and so by induction $\mathcal{R}_{\sigma_j^i}(\tau' \circ \omega)(F^i) = \top$, where $\mathcal{R}_{\sigma_j^i}(\tau' \circ \omega) = \mathcal{R}_{\sigma_j^i \cdot (\omega:C)}(\tau')$ since $\tau'(C) \neq \top$, and so $\mathcal{R}_{\sigma_j^i \cdot (\omega:C)}(\tau')(F^i) = \mathcal{R}_{\sigma_{j+1}^i}(\tau')(F^i) = \top$. In case rule Restore was applied, assuming that σ_j^i had the shape $\sigma \cdot (\omega : C) \cdot \sigma'$, we know that $C \in \varphi_{j+1}^i$, thus every satisfying assignment τ' of φ_{j+1}^i satisfies C. Further, we know from the precondition of Restore that C is clean w.r.t. σ', and so by Lemma 4.3.1, we know that $\mathcal{R}_{\sigma'}(\tau')(C) = \top$. Putting these together, $\mathcal{R}_{\sigma_{j+1}^i}(\tau') = \mathcal{R}_{\sigma\cdot\sigma'}(\tau') = \mathcal{R}_\sigma(\mathcal{R}_{\sigma'}(\tau')) = \mathcal{R}_{\sigma\cdot(\omega:C)}(\mathcal{R}_{\sigma'}(\tau')) = \mathcal{R}_{\sigma\cdot(\omega:C)\cdot\sigma'}(\tau')$, where we know that F^i evaluates to true by induction. Starting a new phase with AddClauses when $0 \leq i < n$ and $j = k_i$ simply means that we extend $\varphi_{k_i}^i$ with a set of clauses Δ_{i+1}. In that case we need to show that $\mathcal{R}_{\sigma_0^{i+1}}(\tau')(F^{i+1}) = \top$

holds for all τ' s.t. $\tau'(\varphi^i_{k_i} \wedge \Delta_{i+1}) = \tau'(\varphi_0^{i+1}) = \top$, where $\mathcal{R}_{\sigma_0^{i+1}}(\tau') = \mathcal{R}_{\sigma^i_{k_i}}(\tau')$ since the rule does not change the reconstruction stack. By the ind. hyp. we know that $\mathcal{R}_{\sigma^i_{k_i}}(\tau')(F^i) = \top$. Further, $F^{i+1} = F^i \wedge \Delta_{i+1}$, where each clause in Δ_{i+1} is clean w.r.t. $\sigma^i_{k_i}$ due to the side condition of AddClauses. Thus, due to Lemma 4.3.1, $\mathcal{R}_{\sigma^i_{k_i}}(\tau')(\Delta_{i+1}) = \mathcal{R}_{\sigma_0^{i+1}}(\tau')(\Delta_{i+1}) = \top$. □

6.3.2 Non-Monotonicity of Solution Reconstruction

In an incremental SAT problem it often happens that some of the new clauses make a previously redundant clause implied. In case this redundant clause was not removed previously, we can simply apply Drop on it when the new clauses are added to the formula. However, if the clause was previously removed via Weaken$^+$, it is stored on the reconstruction stack when the new clauses are added. One could apply Restore and then Drop in order to get rid of it permanently, but this solution might restore more clauses from the stack and thus potentially would increase more than necessary the size of the irredundant clause set. In Figure 6.4 we introduce a new rule, called Flush, that drops implied clauses directly from the reconstruction stack without restoring them.

$$\frac{\varphi\ [\,\rho\,]\ \sigma \cdot (\omega : C) \cdot \sigma'}{\varphi\ [\,\rho\,]\ \sigma \cdot \sigma'}\ \boxed{\xi}$$

Flush

where $\boxed{\xi}$ is $\varphi \models C$

Figure 6.4: Rule to remove implied clauses from the reconstruction stack.

Although this rule intuitively seems trivial, the sad truth is that in our current incremental inprocessing calculus it is not correct to use. The following example demonstrates what could happen if we introduce Flush as it is to our calculus.

Example 6.3.1. *Consider formula* $F = (\neg a \vee b) \wedge (\neg a \vee \neg b) \wedge (a \vee b)$. *Starting the inprocessing from state* $(\neg a \vee b) \wedge (\neg a \vee \neg b) \wedge (a \vee b)\ [\,\emptyset\,]\ \varepsilon$, *clause* $(a \vee b)$ *is implied by* $(\neg a \vee b) \wedge (\neg a \vee \neg b) \wedge (a \vee b)$ *and therefore can be learned, reaching state* $(\neg a \vee b) \wedge (\neg a \vee \neg b) \wedge (a \vee b)\ [\,(a \vee b)\,]\ \varepsilon$. *Since* b *is blocked in clause* $(a \vee b)$, *it can also be eliminated from the irredundant clause set via* Weaken$^+$ *leading to state* $(\neg a \vee b) \wedge (\neg a \vee \neg b)\ [\,(a \vee b)\,]\ (b : (a \vee b))$. *Then the remaining irredundant clauses are blocked by* $\neg a$, *so removing them via* Weaken$^+$ *yields the state* $\emptyset\ [\,(a \vee b)\,]\ (b : (a \vee b)) \cdot (\neg a : (\neg a \vee b)) \cdot (\neg a : (\neg a \vee \neg b))$. *Moving the learned clause* $(a \vee b)$ *into the irredundant set with* Strengthen *leads to the state* $(a \vee b)\ [\,\emptyset\,]\ (b : (a \vee b)) \cdot (\neg a : (\neg a \vee b)) \cdot (\neg a : (\neg a \vee \neg b))$, *where notice that the first clause of* σ *is implied by* φ.

Consider the assignment $\tau = \{a = \top, b = \bot\}$ *that satisfies* φ. *Applying the reconstruction function on it we get* $\mathcal{R}_{(b:(a\vee b))\cdot(\neg a:(\neg a\vee b))\cdot(\neg a:(\neg a\vee\neg b))}(\tau) =$

$\mathcal{R}_{(b:(a\vee b))\cdot(\neg a:(\neg a\vee b))}(\tau) = \mathcal{R}_{(b:(a\vee b))}(\tau \circ \neg a) = \mathcal{R}_{(b:(a\vee b))}(\{a = \bot, b = \bot\}) = \{a = \bot, b = \top\}$. *However, removing* $(b : (a \vee b))$ *from* σ *by* FLUSH *we would get* $\tau \circ \neg a = \{a = \bot, b = \bot\}$ *as result of the reconstruction function, which actually falsifies* $(a \vee b)$.

One possible way to adapt our framework to this new rule is to maintain a stronger form of the reconstruction property. In Def. 6.3.2 we focused on guaranteeing that F is satisfied *after* the complete application of the reconstruction function. And so implicitly it is allowed that the internal steps of solution reconstruction temporarily falsify any clauses in φ or in σ, even if it was already satisfied by previous steps. This implicit possibility is allowed by our published reconstruction property in [94] as well. Below we demonstrate with an example such an internal falsification during solution reconstruction.

Example 6.3.2. *Consider formula* $F = (b \vee c) \wedge (a \vee b)$. *Starting the evaluation from state* $(b \vee c), (a \vee b)\ [\emptyset]\ \varepsilon$, *we can simply just learn the clause* $(a \vee b)$. *Since* $(a \vee b)$ *is blocked on* a *in* φ, *it can be removed via* WEAKEN$^+$, *resulting in the state* $(b \vee c)\ [(a \vee b)]\ (a : (a \vee b))$. *The last irredundant clause* $(b \vee c)$ *is blocked as well so* WEAKEN$^+$ *is applicable again. However, the side condition of* WEAKEN$^+$ *is quite permissive regarding the witnesses of elimination steps, since it must be compatible with a very general notion of redundancy. For example* $\omega = \{c, \neg a\}$ *is a valid witness in that simplification step because* $\omega(b \vee c) = \top$ *and* $\varphi|_{\neg b,\neg c} = \emptyset \models \varphi|_\omega = \emptyset$. *In the resulting state* $\emptyset\ [(a \vee b)]\ (a : (a \vee b)) \cdot (c, \neg a : (b \vee c))$ *rule* STRENGTHEN *can be applied to reach state* $(a \vee b)\ [\emptyset]\ (a : (a \vee b)) \cdot (c, \neg a : (b \vee c))$.

Consider the satisfying truth assignment $\tau = \{a = \top, b = \bot\}$ *of* $(a \vee b)$. *Applying the reconstruction function on it with* $(a : (a \vee b)) \cdot (c, \neg a : (b \vee c))$ *is* $\mathcal{R}_{(a:(a\vee b))\cdot(c,\neg a:(b\vee c))}(\tau)$. *Since* $\tau(b \vee c) \neq \top$, τ *is in the first step updated, resulting in* $\tau' = \{a = \bot, b = \bot, c = \top\}$. *However,* $\tau'(\varphi) = \tau'(a \vee b) \neq \top$.

Though in this example we artificially constructed a witness that will trigger the expected problem, based on the definition of our redundancy property and witnesses, in theory nothing forbids steps like this. So either we need to constrain the concept of witnesses and redundant clauses (that would reduce the generality of our calculus), or we have to consider these possibilities. Based on this observation, the following definition further refines our reconstruction property in order to explicitly prohibit this internal falsification of clauses in φ and in already processed parts of σ, without actually changing the used redundancy property of our framework.

Definition 6.3.3 (Monotone Reconstruction Property)**.** *A state* $\varphi\ [\rho]\ \sigma$ *satisfies the monotone reconstruction property w.r.t. a formula* F *iff for all truth assignments* τ *satisfying* φ, *the result of the reconstruction function* $\mathcal{R}$ *on* τ *and any suffix* σ' *of* σ *is a satisfying assignment for* $\varphi \wedge \sigma'$ *and* $\mathcal{R}_\sigma(\tau)$ *is a satisfying assignment of* F.

The intuition behind this definition is that what once was already satisfied, we do not want to ruin during solution reconstruction, i.e., we want the set of satisfied clauses in $\varphi \wedge \sigma$ to increase monotonically. With the help of this refined reconstruction property we can now construct a provably correct set of inprocessing rules that includes also Flush. However, the price to maintain this property is that we are not allowed to use rule Strengthen (i.e. for Flush we must give up rule Strengthen). The following theorem shows that the resulting calculus satisfies the monotone reconstruction property.

Theorem 6.3.2. *In any derivation starting from the initial state in a calculus consisting of rules* Learn$^-$, Forget, Weaken$^+$, Drop, Restore, AddClauses *and* Flush, *the monotone reconstruction property* $\forall\tau.(\tau(\varphi^i_j) = \top \Rightarrow (\forall\sigma'.\,\mathrm{suffix}(\sigma^i_j,\sigma') \Rightarrow \mathcal{R}_{\sigma'}(\tau)(\varphi^i_j \wedge \sigma') = \top) \wedge \mathcal{R}_{\sigma^i_j}(\tau)(F^i) = \top)$ *holds for each phase* $i = 0 \ldots n$ *and* j *with* $0 \leq j \leq k_i$.

Proof. It is not hard to see that both Prop. 4.4.1 and 4.4.2 of [94] holds in case Flush is added to and Strengthen is removed from our incremental inprocessing calculus defined by Fig. 4.5 in Chapter 4. Thus, we have that $\varphi^i_j \wedge \sigma^i_j \equiv F^i$ and it is enough to show that the reconstruction function on every suffix σ' of the reconstruction stack (including the complete suffix σ^i_j) satisfies $\varphi^i_j \wedge \sigma'$. In the initial state the reconstruction stack is empty thus there is only one possible suffix (ε), and so for any satisfying assignment τ of F^0, $\mathcal{R}_\varepsilon(\tau)(F^0) = \top$. Assume that in a state j of a phase i (where $0 \leq j < k_i$ and $0 \leq i \leq n$), the property holds. In case of rules Learn$^-$, Forget and Drop the reasoning is exactly as in proof of Theorem 6.3.1 and the rule AddClauses does not change the reconstruction stack, thus the previous proof is easily adaptable here.

Our induction hypothesis is that $\forall\tau.(\tau(\varphi^i_j) = \top \Rightarrow (\forall\sigma'.\,\mathrm{suffix}(\sigma^i_j,\sigma') \Rightarrow \mathcal{R}_{\sigma'}(\tau)(\varphi^i_j \wedge \sigma') = \top))$ holds in the current state. If the applied rule is Weaken$^+$, consider an arbitrary truth assignment τ' such that it satisfies $\varphi^i_{j+1} = \varphi^i_j \setminus C$. If there is no such assignment, the property is trivially maintained. Otherwise, we know that any possible suffix θ of $\sigma^i_{j+1} = \sigma^i_j \cdot (\omega : C)$ is either ε or has the form $\sigma' \cdot (\omega : C)$, where σ' is a suffix of σ^i_j. In case θ is ε, by Def. 6.3.1 $\mathcal{R}_\varepsilon(\tau')(\varphi^i_{j+1} \wedge \theta) = \tau'(\varphi^i_{j+1}) = \top$. Otherwise $\mathcal{R}_\theta(\tau') = \mathcal{R}_{\sigma'\cdot(\omega:C)}(\tau')$. Trivially, if $\tau'(C) = \top$, then $\mathcal{R}_{\sigma'\cdot(\omega:C)}(\tau') = \mathcal{R}_{\sigma'}(\tau')$ and then by induction $\mathcal{R}_{\sigma'}(\tau)(\varphi^i_{j+1} \wedge \theta) = \mathcal{R}_{\sigma'}(\tau')(\varphi^i_j \wedge \sigma') = \top$. If $\tau'(C) \neq \top$, then $\mathcal{R}_{\sigma'\cdot(\omega:C)}(\tau') = \mathcal{R}_{\sigma'}(\tau' \circ \omega)$. From the side condition of Weaken$^+$ and from Prop. 4.2.1 together follows $(\tau' \circ \omega)(\varphi^i_{j+1} \wedge C) = \top$ and so by induction $\mathcal{R}_{\sigma'}(\tau' \circ \omega)(\varphi^i_{j+1} \wedge C \wedge \sigma') = \top$, so $\mathcal{R}_\theta(\tau')(\varphi^i_{j+1} \wedge \theta) = \top$.

In case rule Restore is the next step, assume an arbitrary truth assignment τ' s.t. $\tau'(\varphi^i_{j+1}) = \tau'(\varphi^i_j \wedge C) = \top$. Then by induction we know that $\mathcal{R}_\theta(\tau')(\varphi^i_j \wedge \theta) = \top$ holds for all possible suffix θ of $\sigma^i_j = \sigma \cdot (\omega : C) \cdot \sigma'$. Further, due to Lemma 4.3.1, for any suffix θ' of σ' we have that $\mathcal{R}_{\theta'}(\tau')(C) = \top$ since $\tau'(C) = \top$ and C is clean w.r.t. θ'. Every suffix of $\sigma^i_{j+1} = \sigma \cdot \sigma'$ is either a suffix of σ' and thus by induction satisfies the property, or has the form $\theta \cdot \sigma'$ where θ

is a suffix of σ. In that case $\mathcal{R}_{\theta\cdot\sigma'}(\tau') = \mathcal{R}_{\theta}(\mathcal{R}_{\sigma'}(\tau')) = \mathcal{R}_{\theta\cdot(\omega:C)}(\mathcal{R}_{\sigma'}(\tau'))$ since $\mathcal{R}_{\sigma'}(\tau')(C) = \top$. By induction we know that $\mathcal{R}_{\theta\cdot(\omega:C)}(\mathcal{R}_{\sigma'}(\tau')) = \mathcal{R}_{\theta\cdot(\omega:C)\cdot\sigma'}(\tau')$ satisfies $\varphi^i_j \wedge \theta \wedge C \wedge \sigma'$, thus $\mathcal{R}_{\theta\cdot\sigma'}(\tau')(\varphi^i_{j+1} \wedge \theta \wedge \sigma') = \top$ must hold as well.

When rule Flush is employed, the form of σ^i_{j+1} is similar to the case of rule Restore, and thus a similar reasoning is applicable. The only difference that in that case C is not part of φ^i_{j+1}, but $\varphi^i_{j+1} = \varphi^i_j \models C$ due to ξ and since by induction $\mathcal{R}_{\theta'}(\tau)(\varphi^i_{j+1} \wedge \theta') = \top$ for any suffix θ' of σ', we have here as well that $\mathcal{R}_{\sigma'}(\tau)(C) = \top$. □

Note however that most of the currently known and used pre- and inprocessing techniques rely on rule Strengthen and thus it is questionable whether the benefits of Flush outweighs the loss of that rule. Future work should investigate other means to achieve an inprocessing calculus that has monotone reconstruction property and can capture meaningful practical techniques at the same time or whether there are other ways to include rule Flush into our current calculus. The above indicates that the consequences of a stronger reconstruction property could lead to several interesting improvements both in theory and in practice.

6.3.3 Non-Model-Preserving Clause Learning

The current form of rule Learn$^-$ in our incremental calculus is rather strict. The main reason to forbid the learning of non-implied clauses is to somehow guarantee that all the learned clauses can be kept while preserving satisfiability, even when new clauses are added to the formula (see Cor. 4.4.2). The challenging aspect of the general redundancy property that was used as $\sharp$ in the non-incremental calculus, is that this property is not monotone, i.e. if a clause C found to be redundant w.r.t. a formula F, it is not guaranteed that C will be redundant w.r.t. $F \wedge G$ as well, where G is a set of arbitrary new clauses.

The simplest way to overcome this difficulty is to employ a monotone redundancy property, just as we did by changing $\sharp$ to $\triangleright$ in Learn$^-$ in [94]. Another way would be to adopt and adapt the handling of the clause elimination steps to clause learning. Assume we can identify and store the reason (i.e. witness) of each and every learned clause including the order of them. Then in new iterations we could simply retract those clauses (and those that were derived with the help of them) that are not justified any more. For that approach we need to show what kind of information is sufficient to store in order to guarantee soundness and how can we reduce the overhead of this bookkeeping in practice. Further, this approach requires also to consider what is happening when clause learning and elimination are intermixed. Learned clauses can be reasons of eliminations, and so the two techniques must consider each other.

Another possibility to support non-model-preserving clause learning is to somehow guarantee that the redundant, learned clause C does not interfere with any future set of clauses added to the formula. Based on that observation, it is possible to have a special form of non-implied clause learning, for example

as we introduced LEARN$^{\mathcal{D}}$ in Fig. 6.5. This rule allows to learn so called *definition clauses* that can potentially lead to shorter derivations. What is important is to guarantee that we always define a brand new variable (v^* in LEARN$^{\mathcal{D}}$). Notice however that l_1 and l_2 are arbitrary literals, i.e. it can refer even to other definition variables. Beyond that, we have to guarantee that the definition variables do not interfere with any newly added clauses. For that, we mark these variables with *, noting that if a new clause contains v^*, every occurrence of it in φ, ρ and σ needs to be renamed to another fresh variable.

$$\frac{\varphi\ [\,\rho\,]\ \sigma}{\varphi \wedge \mathrm{CNF}(v^* \equiv l_1 \vee l_2)\ [\,\rho\,]\ \sigma}\ \textsc{Learn}^{\mathcal{D}}$$

where $v^* \notin Var(\varphi \wedge \rho \wedge \sigma)$ and $\mathrm{CNF}(v^* \equiv l_1 \vee l_2)$ is clean w.r.t. σ

Figure 6.5: Rule to add definition lemma

Rule LEARN$^{\mathcal{D}}$ allows to simulate extended resolution during incremental inprocessing (see e.g. [10, 123]). However, introducing that rule into our abstract framework would modify most of our invariants and would complicate several otherwise simple concepts. Thus, instead of explicitly introducing it to our calculus, we just point out that these definition clauses can always be added to our problem (more precisely to φ) via ADDCLAUSES (assuming that the uniqueness of the definition variables is guaranteed in advance for each iteration). Doing so, the formula to solve in a new phase becomes $F^{i+1} = F^i \wedge \Delta_{i+1} \wedge E$ in each iteration, where E contains only extension definition clauses, and so $F^i \wedge \Delta_{i+1} \equiv_{sat} F^i \wedge \Delta_{i+1} \wedge E$ holds.

6.3.4 One-Pass Solution Reconstruction

Rule ADDCLAUSES requires to restore a set of clauses from σ and each of these restored clauses may trigger further clauses to be restored, that can trigger again further clauses to restore and so on. Although RESTORE has this recursive manner, it is still sufficient to traverse the reconstruction stack only once, as long as it is done from the proper direction. In our paper we did not have the space to elaborate on why it is so, thus here we give more details.

Rule ADDCLAUSES requires that each new clause $C \in \Delta$ is clean w.r.t. the whole reconstruction stack, i.e. every $(\omega_i : C_i)$ of the stack has to be checked and restored in case $\omega_i \cap \neg C \neq \emptyset$. Notice that the order of these checks does not matter. On the other hand, whenever a witness labelled clause $(\omega_i : C_i)$ of the reconstruction stack needs to be restored, due to $\boxed{\partial}$ we know that every $(\omega_j : C_j) \in \sigma$ where $j > i$ has to be checked and restored in case $\omega_j \cap \neg C_i \neq \emptyset$. In principle we are looking for the smallest set $\mathcal{A}$ of (C, i) clause-index pairs where $C \in (\Delta \cup \sigma)$ is a clause to be added to the formula of the next iteration and $i \in \mathbb{N}_0$ is a position in σ until C does not need to be checked whether

triggers clauses to restore, s.t. (i) $\forall C \in \Delta : (C, 0) \in \mathcal{A}$ (ii) If $(C, i) \in \mathcal{A}$, then $\forall(\omega_j : C_j) \in \sigma : (j > i \wedge \omega_j \cap \neg C \neq \emptyset) \Rightarrow (C_j, j) \in \mathcal{A}$. And so initiating $\mathcal{A}$ with the clauses of Δ (with index 0, since every clause of σ has to be checked), then checking each clause of σ, against the potentially increasing $\mathcal{A}$, starting from $(\omega_1 : C_1)$ to $(\omega_n : C_n)$ we can identify all those clauses that need to be restored and added to φ of the next phase. Those labelled clauses $(\omega_i : C_i)$ that fail the check against the current $\mathcal{A}$ are not just added to $\mathcal{A}$, but also removed from σ.

6.3.5 Scheduling Incremental Inprocessing

An other interesting question that we did not have the space in [94] to address is how to schedule inprocessing when the solver is solving incremental problems. Inprocessing SAT solvers usually have their own rhythm to invoke inprocessing during search (e.g. after certain number of restarts done, conflicts found etc.) and it is limited how much time is allowed to spend on it. In case it is invoked too often or too long, the spent resources outweigh the benefits of simplifications. Thus, usually a schedule of inprocessing has a decreasing frequency of invocation defined by increasing thresholds. However, in an incremental problem sequence using that schedule can lead to extreme cases. When each problem of the sequence is just too simple to reach the first threshold of inprocessing invocation, the formula is never simplified. Another extreme is that the subproblems are really hard and so inprocessing is invoked really frequently at the beginning of each phase. A solution to avoid these extremities in CaDiCaL was to consider the whole incremental problem in the sense of inprocessing scheduling as a stand-alone run. It simply means that in our presented results in [94] none of the relevant inprocessing counters were reset in between phases.

6.3.6 Certificates with Incremental Inprocessing

Here we shortly discuss how proof generation of unsatisfiable formulas is affected by incremental inprocessing. The question is how to handle restored clauses in the produced proofs. Because of the side condition of rule Weaken$^+$, we can be sure that every clause on the reconstruction stack is semantically implied by the original set of clauses (i.e. by F^i). In the current implementation, whenever a clause is restored via Restore, in the proof it is (re-)introduced as an original clause that is part of Δ_{i+1}. In the trivial case it was indeed an input clause that got weakened. However, in case the clause was first learned with Learn$^-$ and then rule Strengthen was applied before it was moved to the stack with Weaken$^+$, the clause is not original. Nevertheless, in that case the clause must be implied (due to the side condition of rule Learn$^-$) and so instead of Weaken$^+$, at some point Drop could have been applied on it. Thus, an inprocessing strategy where rule Drop is preferred over rule Weaken$^+$ in cases when both is applicable avoids these potentially problematic situations.

6.3.7 Reconstructed Solutions after Incremental Inprocessing

The idea behind solution reconstruction at the end of inprocessing is to allow non-model-preserving formula simplifications and at the same time guarantee that a solution for the original, unprocessed formula is provided. However, an implicit side effect of that process is that we lose some of the possible models of the original formula. The reconstructed solution of an inprocessed formula is partially determined by the witnesses of the employed clause elimination steps. Finding different solutions for the simplified formula may result in different reconstructed solutions. Nevertheless, the enumeration of all solutions of the original formula is in most cases impossible if inprocessing was applied.

In the current inprocessing framework we consider always a single witness for every clause elimination step (i.e. when we apply rule WEAKEN$^+$). It would not be hard to redefine (and reimplement) the reconstruction stack such that we can assign more than one witness for an eliminated clause. Of course in that case each of these witnesses should satisfy the side condition of WEAKEN$^+$. Though finding all reasons of redundancy is not necessarily possible, in some cases it might be relatively cheap to identify more than one witnesses (e.g. when more than one literal is blocked in a clause). It could provide benefits not just regarding the number of reconstructible solutions. In case of incremental inprocessing the number of restored clauses depends on how many redundancy reasons are changed due to the new clauses. Having more possible witnesses for an eliminated clause would allow to consider one (or more) of them that is clean w.r.t. the new clauses. Thus in that case instead of restoring the clause, we just need to remove the invalidated witnesses of it. Of course this extension would be beneficial only if the overhead to identify the additional witnesses is negligible or less than the resources consumed by restoring the clauses between iterations. Nevertheless, it is an interesting possibility that might be worth further investigations in the future.

6.4 Duplex Encoding of Staircase At-Most-One Constraint Sets

In this section we present some unpublished details regarding our proposed duplex encoding of at-most-one sequence constraints.

6.4.1 CNF Encoding of BDD Nodes

Given a BDD node with auxiliary Boolean variable b, that decides on variable x_i and has a true child node with variable t and a false child node with variable f, we have several options to represent this node in conjunctive normal form. In [90] an encoding was proposed, where each node of a constructed BDD is encoded simply as an if-then-else node. This encoding (will be referred as `Minisat+`) yields the following four clauses for each node of a BDD:

$$\texttt{Minisat+}:$$
$$(a)\ \ x \wedge t \rightarrow b \qquad (b)\ \ \neg x \wedge f \rightarrow b$$
$$(c)\ \ x \wedge \neg t \rightarrow \neg b \qquad (d)\ \ \neg x \wedge \neg f \rightarrow \neg b.$$

To achieve arc-consistency in the general case, the encoding is extended with the following two clauses in [90] (will be denoted as `GAC-Minisat+`):

$$\texttt{GAC-Minisat+}: \quad (a) - (d) \wedge$$
$$(e)\ \ t \wedge f \rightarrow b \qquad (f)\ \ \neg t \wedge \neg f \rightarrow \neg b.$$

Notice that in our SCAMO constraints these plus clauses are not necessary in order to achieve arc consistency. In each BDD each node is either an AMO or an AMZ node. When an AMO node is encoded, we know that the node output has to be true (i.e. b is a unit clause) and also that the false child has to be true (because it is also an AMO node, i.e. f is a unit clause as well). Thus in that case clauses (e) and (f) are root-satisfied. In case of AMZ nodes, clause (e) is root-satisfied since t is always $\bot$ and due to the same reason the other clause can be simplified by removing t. Regarding clause (f), it is easy to see that each decision variable x_i belongs not just to the current AMZ node, but to an AMO node as well, in a clause $\neg x_i \vee t$ where t is actually the false child of the current AMZ node. Thus when $\neg f$ holds, the AMO node can propagate $\neg x_i$, and so $\neg f \wedge \neg x_i$ together propagate $\neg b$ via clause (d). So even though in a longer way, but the propagation strength of `Minisat+` and `GAC-Minisat+` is the same in our context, due to the structure of the BDDs and the existence of the bonding clauses.

Another possible translation of pseudo-Boolean constraints to clausal form via BDDs was proposed in [2] (we will denote this encoding simply as `BDD-1`). In this work the monotonicity of the pseudo-Boolean constraints is exploited

during encoding, resulting in only the following two clauses:

`BDD-1` :

$$(g)\quad \neg f \rightarrow \neg b \qquad\qquad (h)\quad x \wedge \neg t \rightarrow \neg b.$$

Since our problem consists of AMO and AMZ constraints that are both monotonically decreasing, `BDD-1` as arc consistent encoding is applicable. Notice that clause (c) of `Minisat+` occurs as clause (h) in `BDD-1`. Further, in case of AMZ nodes, clauses (f) and (g) become identical.

In our experiments, presented in Chapter 5, we combined the `Minisat+` and the `BDD-1` encodings (will denote this encoding as `CPAIOR`). More precisely, we encoded the AMO nodes with `BDD-1`, but most importantly we did not simplify this encoding, i.e. we kept all the root satisfied clauses and falsified literals. In case of AMOs over less than three variables the second clause was omitted. For AMZ nodes we employed `Minisat+` with simplifications.

Table 6.5 summarizes the presented encodings of each BDD node in case of SCAMO constraints by the resulting clauses. To emphasize the similarities between the encodings, the clauses are sorted in each column following the same order. Notice that each encoding has an equivalent CNF for AMO nodes (although `CPAIOR` is kept without simplifications) while they are noticeably different for the AMZ nodes.

Table 6.5: Different clausal representations of AMO and AMZ nodes.

	`CPAIOR`	`Minisat+`	`GAC-Minisat+`	`BDD-1`
`AMO`	$(\neg b \vee \neg x \vee t)$ $(\neg b \vee f)$	$(\neg x \vee t)$ –	$(\neg x \vee t)$ –	$(\neg x \vee t)$ –
`AMZ`	$(\neg b \vee \neg x)$ $(\neg b \vee x \vee f)$ $(b \vee x \vee \neg f)$ –	$(\neg b \vee \neg x)$ $(\neg b \vee x \vee f)$ $(b \vee x \vee \neg f)$ –	$(\neg b \vee \neg x)$ $(\neg b \vee x \vee f)$ $(b \vee x \vee \neg f)$ $(\neg b \vee f)$	$(\neg b \vee \neg x)$ – – $(\neg b \vee f)$

Though each of these encodings is arc consistent in case the node variables of the corresponding constraints are forced to be true with unit clauses (see [2, 90] and our previous reasoning), they perform differently, as Table 6.6 shows. Considering the same 24 antibandwidth problems as in [97], the table reports the number of solved queries and the required time and memory consumption for it in case of each BDD node encoding.

Although `CPAIOR` and `Minisat+` are logically identical, having root-satisfied clauses and root-falsified literals somehow made `CPAIOR` slightly better on small instances (upper table). However, on larger problems (lower table) `Minisat+` can solve more formulas. A possible explanation of this difference is simply the order of clauses and variables in the two encodings (see e.g. [42] on possible consequences of formula scrambling).

Table 6.6: Results of different node encodings to solve the antibandwidth problem (TO = 1800 seconds and MO = 7 GB).

INSTANCE	A	B	C	D	E	F	G	H	I	J	K	L
$\|V\|$	30	32	39	48	49	54	57	59	85	100	117	118
$\|E\|$	103	90	46	176	59	124	127	281	219	247	162	179
LB	6	9	16	8	21	12	12	8	19	32	46	39
UB	8	9	17	9	22	13	14	8	27	40	58	39
	6	**9**	**17**	**9**	**21**	**13**	**13**	**8**	**23**	**35**	**49**	**39**
CPAIOR	188.69 s	1.37 s	3.92 s	0.35 s	3.38 s	1.35 s	0.55 s	0.48 s	TO	592.57 s	TO	1.02 s
	44 MB	11 MB	14 MB	14 MB	13 MB	19 MB	17 MB	21 MB	299 MB	203 MB	488 MB	53 MB
	6	**9**	**17**	**9**	**21**	**13**	**13**	**8**	22	35	**49**	**39**
Minisat+	85.1 s	1.38 s	23.87 s	0.27 s	8.91 s	0.68 s	0.81 s	0.29 s	TO	TO	TO	0.5 s
	38 MB	11 MB	36 MB	14 MB	23 MB	16 MB	21 MB	21 MB	710 MB	461 MB	461 MB	49 MB
	6	**9**	**17**	**9**	**21**	**13**	**13**	**8**	22	**35**	48	**39**
GAC-Minisat+	132.77 s	0.45 s	9.67 s	0.44 s	3.14 s	0.36 s	1.95 s	0.23 s	TO	645.47 s	TO	0.97 s
	46 MB	9 MB	20 MB	14 MB	13 MB	14 MB	21 MB	24 MB	383 MB	154 MB	400 MB	54 MB
	6	**9**	**17**	8	21	12	12	-	-	-	-	**39**
BDD-1	179.45 s	34.64 s	32.25 s	TO	TO	TO	TO	TO	TO	TO	TO	1793.49 s
	56 MB	38 MB	42 MB	290 MB	323 MB	357 MB	433 MB	536 MB	573 MB	764 MB	823 MB	949 MB

INSTANCE	M	N	O	P	Q	R	S	T	U	V	W	X
$\|V\|$	420	420	425	445	494	503	546	592	662	675	685	715
$\|E\|$	3720	3720	1267	1682	586	2762	1341	2256	906	1290	1282	2975
LB	28	28	91	78	219	46	256	103	219	326	136	112
UB	72	72	173	120	246	71	272	150	220	337	136	142
	34	**34**	99	-	-	**62**	-	-	**220**	-	**136**	-
CPAIOR	TO	TO	TO	TO	TO	TO	TO	TO	328.84 s	TO	14.35 s	TO
	1558 MB	1558 MB	1048 MB	1581 MB	1033 MB	1685 MB	1120 MB	2252 MB	1584 MB	1508 MB	1418 MB	3400 MB
	34	**34**	**100**	**78**	-	**62**	-	-	**220**	-	**136**	-
Minisat+	TO	TO	TO	TO	TO	TO	TO	TO	35.24 s	TO	14.03 s	TO
	1491 MB	1491 MB	943 MB	1582 MB	1023 MB	1571 MB	1103 MB	1992 MB	1459 MB	1540 MB	1349 MB	2691 MB
	34	**34**	97	-	-	60	-	-	**220**	-	**136**	**112**
GAC-Minisat+	TO	TO	TO	TO	TO	TO	TO	TO	185.06 s	TO	22.16 s	TO
	1486 MB	1486 MB	1212 MB	1662 MB	1035 MB	1683 MB	1150 MB	2596 MB	1551 MB	1511 MB	1838 MB	2803 MB
	-	-	-	-	-	-	-	-	-	-	-	-
BDD-1	TO	TO	TO	TO	TO	TO	TO	TO	TO	TO	TO	TO
	3094 MB	3094 MB	2834 MB	2036 MB	3021 MB	2304 MB	2337 MB	3013 MB	3119 MB	3481 MB	2616 MB	3257 MB

There is no large difference between the two encodings proposed in [90] in our context. All in all, although `GAC-Minisat+` can solve two formulas where `Minisat+` fails (in instances J and X), in general `Minisat+` seems to be the slightly better choice. Though encoding BDDs via `BDD-1` leads to the smallest amount of clauses, the memory consumption of the problem solving was here the largest. All in all the performance of that approach was relatively poor in our application since it failed on several smaller instances already, where the other encodings led immediately to an answer (see e.g. the instances D-G).

6.4.2 Arc consistency of Duplex Encoding

As it was pointed out in [97], the arc consistency of our proposed duplex encoding completely depends on the arc consistency of the employed CNF encoding of each constructed BDDs.

Consider an AMO constraint $x_1 + \cdots + x_n \leq 1$ and a BDD built for it via the method `BDD-AMO`$(\langle x_1 \cdots x_n \rangle)$. Let $\mathcal{B}$ contain the clauses of each node of that BDD (considering e.g. the clauses of the second column in Table 6.5). Then we know that the formula $\mathcal{B} \wedge b_1$, where b_1 is the auxiliary Boolean variable that was introduced for the root node of this AMO BDD, restores generalized arc consistency through unit propagation. Thus, unit propagation on $\mathcal{B} \wedge b_1 \wedge x_i$ for any $1 \leq i \leq n$ would assign false to every other variable x_j of that constraint. Similarly, given an AMZ constraint $x_1 + \cdots + x_n \leq 0$ and the encoding of the related BDD as clauses in $\mathcal{B}$, we can assume that unit propagation on $\mathcal{B} \wedge b_0$, where b_0 is the root of that BDD, propagates every variable x_i $(1 \leq i \leq n)$ to be false. From that it also follows that unit propagation on $\mathcal{B} \wedge x_i$ forces b_0 to be false.

Now we can more formally reason about the arc consistency of our duplex encoding. Consider a set of variables $X = \langle x_1\, x_2 \cdots x_n \rangle$ and a staircase at-most-one constraint set $\mathcal{C}$ with width w s.t. $2 < w \leq n$, i.e. $\mathcal{C} = \text{SCAMO}(\text{X}, \text{w})$.

Theorem 6.4.1. *Assuming an arc consistent encoding for each node of the constructed BDDs, our proposed duplex encoding of $\mathcal{C}$ restores generalized arc consistency through unit propagation.*

Proof. Let $\mathcal{D}$ be the set of clauses resulting from the duplex encoding of $\mathcal{C}$, consisting of the clauses of each AMO and AMZ BDDs together with the bonding clauses. We show that given a partial truth assignment to the variables in X, if assigning a variable x_i to true would violate $\mathcal{C}$, unit propagation assigns false to x_i through the clauses in $\mathcal{D}$.

Assigning a variable x_i to true can violate $\mathcal{C}$ if and only if there is already a variable x_j assigned to true such that $|i - j| < w$. Then, there are two possibilities: either x_i and x_j belong to the same window ω_k, or to two consecutive windows ω_k, ω_{k+1} during duplex encoding.

The first case is trivial, since in that case we have a forward BDD for the AMO constraint over the variables in ω_k and an arc consistent encoding of each node

of it in $\mathcal{D}$. Beyond that, in the bonding clauses we have a unit clause ω_k^f-l_1-AMO that enforces the root of that BDD to be true. Thus, unit propagation on $\mathcal{D} \wedge x_j$ enforces every other variable in ω_k (including x_i) to be false. In case x_i and x_j belong to two consecutive windows ω_k, ω_{k+1}, we need to take a closer look on the AMZ constraints. W.l.o.g. assume that $j < i$, i.e. x_j belongs to the window ω_k, while x_i is in ω_{k+1}. Then, since x_j is true, the variable representing the root node of the AMZ constraint over the variables $\langle x_j \cdots x_{k_\omega} \rangle$ must be false. Thus, $\neg\omega_k^f$-l_t-AMZ holds, where l_t is the layer of x_j in window ω_k (i.e. x_j is the tth element in ω_k). Then the binary clause (ω_k^f-l_t-AMZ$\vee\omega_{k+1}^b$-$l_{(w-t)+2}$-AMZ) of the bonding clauses will propagate ω_{k+1}^b-$l_{(w-t)+2}$-AMZ, that forces every variable in the $t-1$ long prefix of window ω_{k+1} to be false. Since $i - j < w$, we know that it includes x_i as well. □

6.4.3 Size of Duplex Encoding

In our submitted paper we provided a very generous upper bound on the number of necessary clauses to encode a complete SCAMO constraint set over n variables with width w. The presented upper bound was calculated with an encoding where the AMO nodes may have up to two clauses, the last window is assumed to have width w as well and we assumed that both the first and the last windows are encoded in both directions.

In reality, assuming that the BDD nodes are encoded for example as in `Minisat+` (see Section 6.4.1), a more realistic approximation of the number of clauses would be as follows:

$$\begin{aligned}
\#\text{BDD-clauses} &\approx ((2 \cdot (M-1) - 1) \cdot 4 \cdot (w-1)) + 4 \cdot (w' - 1) \\
&= 4 \cdot (2M \cdot (w-1) - 3 \cdot w + w' + 2) \\
\#\text{BOND-clauses} &\approx (M-2) \cdot (3 \cdot (w-1) - 2) + 3 \cdot w' - 2 + 2 \cdot (M-1) \\
&= 3 \cdot M \cdot w + M - 6 \cdot w + 3 \cdot w' - 2 \\
\#\text{BDD} + \#\text{BOND-clauses} &\approx 11 \cdot M \cdot w - 7 \cdot M - 18 \cdot w + 7 \cdot w' + 6
\end{aligned}$$

where $w' \in \{1, \ldots, w\}$ is the width of the last window. We call it approximation, since to keep the implementation simpler, some of the unit clauses might be added more than once. Further, it is a design choice to use the same variables or introduce equivalences between root-nodes of the constructed forward and backward BDDs. In our implementation we added $4 \cdot (M-2)$ binary clauses to express this equivalence. Figure 6.6 depicts how large is actually the difference between our provided upper bound in [97] (increased with $4 \cdot (M-2)$) and the more accurate number of clauses presented here, in the case of $n = 500$.

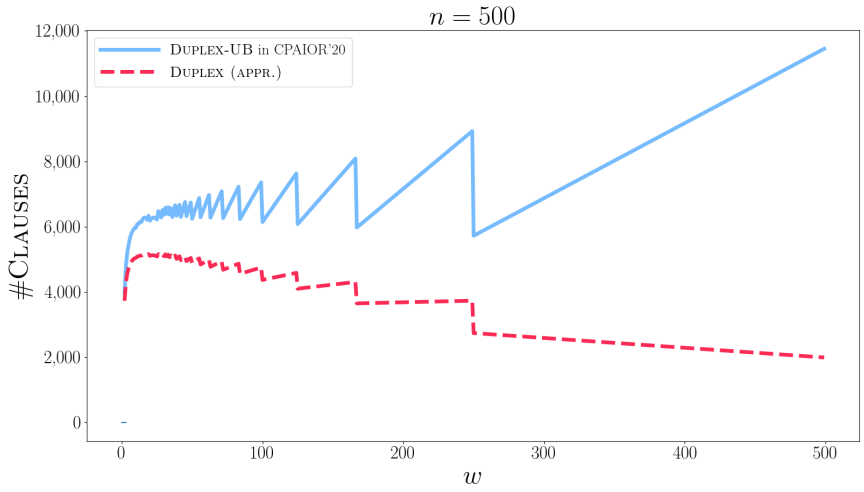


Figure 6.6: Comparison of number of clauses for encoding a single SCAMO constraint set on $n = 500$ variables and width w between 2 and 500.

6.4.4 Strategies to Solve the Antibandwidth Problem

Our approach to solve the antibandwidth problem in Chapter 5 was an iterative method. In each iteration we asked a SAT solver whether it is possible to assign unique labels to each node of the given graph, such that the smallest difference between labels of neighbours is at least w, where w then was used as a width in the SCAMO encoding. If the constructed formula was satisfiable, w was increased with one and a new formula was built to solve. In case the formula was unsatisfiable, it proved that the previous w was the maximal possible minimum difference, i.e. the antibandwidth of that graph. From previous work and theoretical results, for each graph we had a lower and an upper bound for the possible antibandwidth values.

This search method in principle follows the pattern of iterative MaxSAT solvers (see e.g. [182]), where a linear search is performed to identify the maximal number of satisfiable clauses. Sticking with that parallel drawn between iterative MaxSAT solving algorithms and our antibandwidth search, it is not hard to see that our iteration could have started from the upper bound of the antibandwidth value or a binary search between the bounds would have been possible as well. CaDiCaL, the SAT solver we used in our experiments, was the best performing solver on the satisfiable instances of the SAT race in 2019 [122]. Thus, it is not surprising that the best results were found with the presented increasing search, where all formulas given to the SAT solver, but the last one, must be satisfiable (referred as "Sat-Unsat" search in MaxSAT context). For the sake of completeness, and to see if there were consequences of this choice, we implemented and repeated the experiments with the other two strategies and present the results in Table 6.7.

Table 6.7: Results of different strategies to solve the antibandwidth problem (TO = 1800 seconds and MO = 120 GB).

Instance	$\|V\|$	$\|E\|$	LB	UB	Duplex – W-Increasing			Duplex – W-Decreasing			Duplex – Binary Search		
					Obj.	Time	MB	Obj.	Time	MB	Obj.	Time	MB
A-pores_1	30	103	6	8	**6**	185.52	52	**6**	189.8	47	**6**	192.68	47
B-ibm32	32	90	9	9	**9**	1.3	11	**9**	1.31	11	**9**	1.3	11
C-bcspwr01	39	46	16	17	**17**	3.85	13	**17**	3.7	14	**17**	3.96	13
D-bcsstk01	48	176	8	9	**9**	0.25	14	**9**	0.16	13	**9**	0.27	14
E-bcspwr02	49	59	21	22	**21**	3.37	13	**21**	3.55	13	**21**	3.49	13
F-curtis54	54	124	12	13	**13**	1.33	18	**13**	1.35	18	**13**	1.4	18
G-will57	57	127	12	14	**13**	0.57	19	**13**	0.23	17	**13**	0.24	15
H-impcol_b	59	281	8	8	**8**	0.54	22	**8**	0.66	22	**8**	0.46	22
I-ash85	85	219	19	27	$23 \leq$	TO	331	< 26	TO	137	$23 \leq$	TO	125
J-nos4	100	247	32	40	**35**	585.33	190	**35**	588.04	163	**35**	610.62	179
K-dwt__234	117	162	46	58	$49 \leq$	TO	477	< 56	TO	152	-	TO	250
L-bcspwr03	118	179	39	39	**39**	0.99	58	**39**	1.15	54	**39**	1.19	54
M-bcsstk06	420	3720	28	72	$34 \leq$	TO	1621	< 46	TO	1481	< 50	TO	1500
N-bcsstk07	420	3720	28	72	$34 \leq$	TO	1621	< 45	TO	1480	< 50	TO	1500
O-impcol_d	425	1267	91	173	$99 \leq$	TO	1043	< 139	TO	827	-	TO	1022
P-can__445	445	1682	78	120	-	TO	1581	< 111	TO	1057	-	TO	1199
Q-494_bus	494	586	219	246	-	TO	1167	< 244	TO	959	-	TO	949
R-dwt__503	503	2762	46	71	$62 \leq$	TO	1680	-	TO	1561	$58 \leq$	TO	1634
S-sherman4	546	1341	256	272	-	TO	1129	< 271	TO	1131	-	TO	1093
T-dwt__592	592	2256	103	150	-	TO	2253	< 148	TO	1755	-	TO	2170
U-662_bus	662	906	219	220	**220**	319.73	1564	**220**	251.07	1565	**220**	340.25	1542
V-nos6	675	1290	326	337	-	TO	1571	< 336	TO	1624	-	TO	1559
W-685_bus	685	1282	136	136	**136**	14.33	1428	**136**	16.45	1453	**136**	15.51	1418
X-can__715	715	2975	112	142	-	TO	3312	-	TO	2529	-	TO	2748

First of all, we can see that all three strategies could solve optimally exactly the same set of instances. For most of the unsolved problem instances the "Unsat-Sat" direction of search (noted as `W-Decreasing` in Table 6.7) successfully refined the upper bounds several times. In contrast to that, the binary search either found the optimal value or could not refine any of the bounds significantly. If we would construct a virtual best solver result, where we consider the tightest lower and upper bounds found by any of the methods, the binary search would not contribute on any of the unsolved instances. On the instances where we found the antibandwidth, we observed that the SAT queries became harder as the width got closer to the optimal value. This observation might explain why binary search is not an efficient approach for this problem.

6.4.5 Benchmark Submission to the SAT Competition

Our experimental results in Chapter 5 showed that although our duplex encoding yields smaller formulas than alternative SAT encodings, the resulting problem instances are still rather challenging for current solvers. Another tendency that we observed during the experiments is that for each graph the closer the width of the SCAMO constraint to the optimal antibandwidth was, the longer it took to solve the resulting SAT formula. Further, for each graph we know that a SAT formula of a SCAMO constraint with a width that is smaller or equal than the antibandwidth, must be satisfiable, while formulas with widths higher are unsatisfiable. Thus, all in all, each larger graph in our experiment is actually a perfect source to generate a sequence of interesting SAT formulas where both the "difficulty" and the satisfiability of the formula can be influenced.

In the annual competition of SAT solvers (see e.g. [121, 122]) a formula is considered "interesting" if it requires more than a minute to be solved with `MiniSAT` [88], but takes less than an hour to solve with a current SAT solver. We generated for each graph a sequence of interesting SAT formulas based on that definition. More precisely, for each graph we considered every consecutive widths in a wide range around the antibandwidth, or around the width $(\mathrm{LB} + \mathrm{UB})/2$ in case the antibandwidth was not known, and generated our duplex encoding of the feasibility query with each of these widths. Then we tried to solve each of the generated formulas with `MiniSAT` in less than a minute. After that, we dropped all those formulas where `MiniSAT` was successful and invoked `CaDiCaL` on the remaining ones, allowing one hour solving time. In the end, we identified 121 interesting problems (91 satisfiable and 30 unsatisfiable).

Figure 6.7 shows the performance on a cactus plot of CaDiCaL (version 0v8) on these instances. Due to the initial filtering with Minisat, the required solving times of the resulting set of problem instances range between few seconds and one hour with a very balanced distribution. On the figure the color of the markers indicate whether the answer was SAT (green) or UNSAT (dark red) and the letter of the marker indicates which graph was the base of the problem. We can see that the unsatisfiable problems were relatively easier to solve and

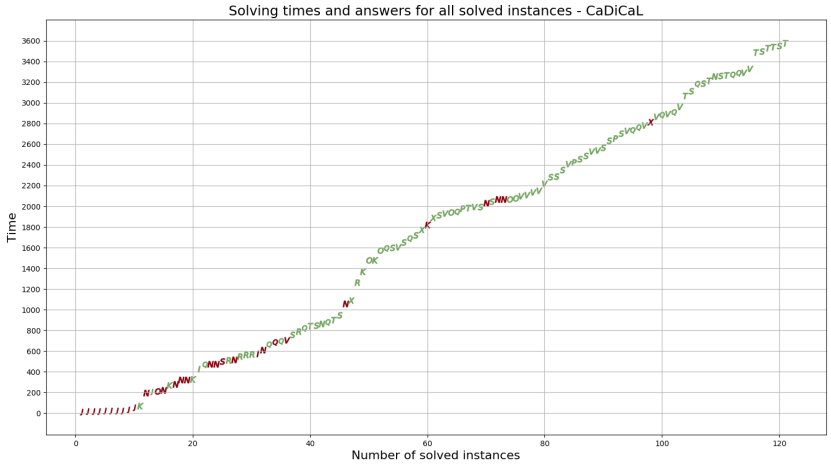


Figure 6.7: Required solving times of problem instances generated from duplex encoding of the antibandwidth problem (CaDiCaL version 0v8).

the hardest solved instances came from the two larger graphs marked with "T" and "S". We submitted these 121 instances together with some further unsolved problems as a benchmark set to the SAT competition 2020.

Chapter 7

Conclusion

In this thesis we presented several SAT-based solution methods for computational problems that are beyond SAT. Each presented method either extends and adapts existing SAT solving techniques, or simply exploits the strengths of these solvers in a combined procedure. This chapter summarizes our findings, focusing on our main contributions and raising some still unanswered questions.

7.1 Thesis Contributions

The main contributions of this thesis can be organized around the computational problems that they are concerned with.

- Regarding quantified Boolean formulas, we introduced a new abstract calculus that precisely captures the behaviour of duality-aware search-based QBF solvers and allows to reason about these systems. We showed potential benefits of a symmetric problem formulation where the DNF representation is complete (e.g. searching for pure literals in cubes). Further, we shortly presented how to adapt our calculus to reason about search-based QBF solvers without the duality property. Beyond that, we informally introduced a new procedure where search-based QBF solving is combined with theory reasoning. In some preliminary experiments we showed how this procedure can be employed as an incomplete solution method to problems with quantified Boolean variables and equalities.

- In the context of maximum satisfiability w.r.t. background theories, we introduced an abstract solver to capture the behaviour of an implicit hitting set based MaxSMT solver. The IHS approach is one of the most successful approaches to tackle MaxSAT problems. To the best of our knowledge, our system is the first attempt to lift this technique to the context of MaxSMT. The main benefit of our proposed solution method is the clear separation of theory reasoning, optimization and Boolean reasoning; thereby allowing to exploit more specialized procedures for each component. As a further contribution, we provided an abstract formalization for assumption based SAT solving in our calculus. From the practical perspective, the experiments with our implementation showed the potential of the introduced flexible framework.

- Incremental SAT solving is essential in many application domains where these solvers are employed. Inprocessing is another technique that significantly contributes to the success of these tools. Though in the past there were attempts to efficiently combine these techniques, our presented solution is so-far the most general approach. We introduced a simple, elegant abstraction of incremental inprocessing, by altering and extending existing work on non-incremental inprocessing. While previous work on incremental inprocessing focused on specific simplification techniques one-by-one, our framework is based on a general redundancy property that embraces every currently used inprocessing technique. Although the resulting system introduces some limitations regarding learned clauses, it still allows most of the practical formula simplifications to be combined with incremental solving. Another important innovation of our technique is that it completely automates inprocessing of incremental problems. Our method eliminates the error-prone burden for the user to identify parts of the problems that might change in later iterations. Further, our proposed algorithm is very simple to implement in any inprocessing SAT solver, thus it can be employed easily in other systems. All in all, we believe that our introduced solution method significantly contributed to the state-of-the-art in incremental SAT solving.

- In our last work we proposed a combined solution procedure for the antibandwidth optimization problem. Exploiting SAT solvers in solving problems that are usually tackled with constraint or integer programming approaches is a very interesting research direction that potentially leads to efficient hybrid solution methods in the long run. Our main contribution in that work was a BDD-based encoding of at-most-one sequence constraints. Sequence constraints are common building blocks of problems in constraint programming. One novelty of our proposed encoding is that we construct multiple BDDs with different variable orderings for each decomposed constraint. While any previous off-the-shelf SAT encoding of these constraints is at least quadratic, our proposed solution leads to an efficient structure sharing and thereby provides a problem representation with linear size w.r.t. the number of variables. Our experiments on the antibandwidth problem showed the benefits of this compact encoding.

7.2 Author Contributions

All the papers presented in Chapters 2-5 are results of a collaborative effort. The works could not have happened by individual attempts and the influence of co-authors makes it hard to attribute any contributions solely to a single author. Nevertheless, here we try to isolate contributions made solely by the author of this thesis, as it is required. Note however that most of these contributions were discussed and refined in collaboration, thus in all cases the final results are the

improved outcomes of a joint work.

Chapter 2 [96] Combining theory reasoning with QBF solving in order to address synthesis problems was proposed by the present author. The main motivation for the paper presented in Chapter 2 was to provide a first step towards a formalization where this is possible. The present author constructed the very first (not yet correct) draft of the proposed calculus and worked out examples to demonstrate possible derivations.

Chapter 3 [93] Solving MaxSAT with the IHS approach was introduced in a previous work [79] co-authored by Fahiem Bacchus. The incentive to involve theory reasoning in IHS based MaxSAT solving to address MaxSMT was given by the present author. Further, the present author implemented two different instantiations of the introduced abstract MaxSMT solver, provided the operational description of possible instantiations in the paper, generated the scalable problem benchmarks and conducted the experiments.

Chapter 4 [94] This work closes an important gap in the theory of previously introduced work [138] co-authored by Armin Biere, by adapting and extending the inprocessing framework such that it is now applicable to incremental problems. The present author constructed the main proofs regarding the correctness of the introduced incremental calculus, described the inference rules, constructed the examples used in the paper, and proposed the algorithm for the implementation.

Chapter 5 [97] This work exploits and adapts the iterative MIP formalization of the antibandwidth problem that was introduced by Markus Sinnl in [220]. The present author proposed to consider each constraint twice, once in a right and once in a left associative way. Further, the present author implemented and provided the description of the encoding in the paper, analysed the size and arc consistency of the new representation, developed the encoding tool to solve the antibandwidth problem and conducted the experiments.

Chapter 6 The extensions and experiments presented in Chapter 6 were written and conducted by the present author based on previously found or discussed joint results. The algorithm of `QBeq` was proposed by the present author, while it was implemented in a collaboration with Florian Lonsing.

A further paper where the present author was a collaborating co-author is [95]. In that paper we showed how Skolem-function reconstruction from partial certificates of preprocessed QBFs can be achieved. Though subject-wise this work is related to our discussions here, the present author was not a main contributor, thus it is not included in that thesis.

7.3 Future Work

Most of the computational problems discussed in this thesis are studied since several years and will most likely remain studied for a while still. There are some natural continuations of our presented work that we describe here shortly.

For instance, combining theory reasoning with QBF evaluation is yet completely in its infancy. Several open questions were raised in Section 6.1.9 that must be answered in order to see whether it is an interesting decision procedure.

Regarding the combination of theory reasoning with maximum satisfiability, a formal framework to describe core-guided and iterative solution approaches is still missing. The goal would be to gain a calculus where all different approaches are captured in a unified way. Such a system could help to understand the strengths and limits of the different approaches.

As we saw in Chapter 4 and in Section 6.3, our proposed solution for incremental inprocessing prohibits some inprocessing techniques that are important from the practical perspective (e.g. blocked clause addition or symmetry breaking). Allowing these techniques could further improve incremental SAT solvers. Another important question in incremental inprocessing that was not addressed in our calculus is the role of assumptions. Handling them as root-level decisions forbids to consider them during inprocessing. Exploiting their presence in a general, efficient and sound way would lead to smaller formulas during inprocessing and thus it is definitely something that is worth further investigation.

Adapting our antibandwidth solution method to the bandwidth problem is relatively straightforward, but remains future work. Our proposed BDD-based encoding at several points exploits the fact that we consider only at-most-one constraints. It would be interesting to see whether our encoding could be adapted to gain a compact representation of at-most-k sequence constraints. Another potential direction of further research concerns combining multiple variable orders in decision diagrams. Decision diagrams are frequently used in symbolic optimization. In most cases the diagrams are either relaxed, so that they can provide bounds, or restricted so that they can provide a heuristic. In our work we saw that constructing several BDDs simultaneously over the same problem with different variable orderings led to a compact problem representation. The question is whether there is a potential benefit of that idea in the context of the decision diagrams in symbolic optimization.

There are many practical and theoretically challenging computational problems that are related to SAT, either by extending or by exploiting it. In this thesis we had a glance only on a very few of them, while many more remain intriguing subjects of future research.

Bibliography

[1] Ignasi Abío, Graeme Gange, Valentin Mayer-Eichberger, and Peter J. Stuckey. On CNF encodings of decision diagrams. In *13th International Conference on Integration of AI and OR Techniques in Constraint Programming (CPAIOR)*, volume 9676 of *LNCS*, pages 1–17. Springer, 2016.

[2] Ignasi Abío, Robert Nieuwenhuis, Albert Oliveras, Enric Rodríguez-Carbonell, and Valentin Mayer-Eichberger. A new look at BDDs for pseudo-Boolean constraints. *J. Artif. Intell. Res.*, 45:443–480, 2012.

[3] Tobias Achterberg. SCIP: solving constraint integer programs. *Math. Program. Comput.*, 1(1):1–41, 2009.

[4] Wilhelm Ackermann. *Solvable cases of the Decision Problem*. Studies in Logic and the Foundations of Mathematics, North-Holland, Amsterdam,, 1954.

[5] Carlos Ansótegui, Fahiem Bacchus, Matti Järvisalo, and Ruben Martins. MaxSAT Evaluation 2017, 2017. `http://mse17.cs.helsinki.fi/`.

[6] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. SAT-based MaxSAT algorithms. *Artif. Intell.*, 196:77–105, 2013.

[7] Carlos Ansótegui, Joel Gabàs, and Jordi Levy. Exploiting subproblem optimization in SAT-based MaxSAT algorithms. *J. Heuristics*, 22(1):1–53, 2016.

[8] Josep Argelich, Daniel Le Berre, Inês Lynce, João Marques-Silva, and Pascal Rapicault. Solving linux upgradeability problems using Boolean optimization. In *1st International Workshop on Logics for Component Configuration, (LoCoCo)*, volume 29 of *EPTCS*, pages 11–22, 2010.

[9] Bengt Aspvall, Michael F. Plass, and Robert Endre Tarjan. A linear-time algorithm for testing the truth of certain quantified Boolean formulas. *Information Processing Letters*, 8(3):121–123, 1979.

[10] Gilles Audemard, George Katsirelos, and Laurent Simon. A restriction of extended resolution for clause learning SAT solvers. In *24th Conference on Artificial Intelligence (AAAI)*. AAAI Press, 2010.

[11] Gilles Audemard, Jean-Marie Lagniez, and Laurent Simon. Improving Glucose for incremental SAT solving with assumptions: Application to MUS extraction. In *16th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 7962 of *LNCS*, pages 309–317. Springer, 2013.

[12] Gilles Audemard and Laurent Simon. Glucose and Syrup in the SAT'17. In Tomáš Balyo, Marijn J. H. Heule, and Matti Järvisalo, editors, *Proc. of SAT Competition 2017 – Solver and Benchmark Descriptions*, volume B-2017-1 of *Department of Computer Science Series of Publications B*, pages 16–17. University of Helsinki, 2017.

[13] Abdelwaheb Ayari and David A. Basin. QUBOS: deciding quantified Boolean logic using propositional satisfiability solvers. In *4th International Conference on Formal Methods in Computer-Aided Design (FMCAD)*, volume 2517 of *LNCS*, pages 187–201. Springer, 2002.

[14] Fahiem Bacchus. GAC via unit propagation. In *13th International Conference on Principles and Practice of Constraint Programming (CP)*, volume 4741 of *LNCS*, pages 133–147. Springer, 2007.

[15] Fahiem Bacchus and Matti Järvisalo. Algorithms for maximum satisfiability with applications to AI. AAAI-16 Tutoral `https://www.cs.helsinki.fi/group/coreo/aaai16-tutorial/`.

[16] Fahiem Bacchus and Jonathan Winter. Effective preprocessing with hyper-resolution and equality reduction. In *Selected Revised Papers of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 2919 of *LNCS*, pages 341–355. Springer, 2003.

[17] John Backes, Pauline Bolignano, Byron Cook, Catherine Dodge, Andrew Gacek, Kasper Søe Luckow, Neha Rungta, Oksana Tkachuk, and Carsten Varming. Semantic-based automated reasoning for AWS access policies using SMT. In *18th International Conference on Formal Methods in Computer-Aided Design (FMCAD)*, pages 1–9. IEEE, 2018.

[18] Valeriy Balabanov and Jie-Hong R. Jiang. Unified QBF certification and its applications. *Formal Methods in System Design*, 41(1):45–65, 2012.

[19] Tomáš Balyo, Armin Biere, Markus Iser, and Carsten Sinz. SAT Race 2015. *Artificial Intelligence*, 241:45–65, 2016.

[20] 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 *17th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 8561 of *LNCS*, pages 317–332. Springer, 2014.

[21] Tomáš Balyo, Marijn J. H. Heule, and Matti Järvisalo, editors. *Proc. of SAT Competition 2016 – Solver and Benchmark Descriptions*, volume B-2016-1 of *Department of Computer Science Series of Publications B*. University of Helsinki, 2016.

[22] Tomáš Balyo, Marijn J. H. Heule, and Matti Järvisalo, editors. *Proc. of SAT Competition 2017 – Solver and Benchmark Descriptions*, volume B-2017-1 of *Department of Computer Science Series of Publications B*. University of Helsinki, 2017.

[23] Richa Bansal and Kamal Srivastava. Memetic algorithm for the antibandwidth maximization problem. *J. Heuristics*, 17(1):39–60, 2011.

[24] Haniel Barbosa, Pascal Fontaine, and Andrew Reynolds. Congruence closure with free variables. In *23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 10206 of *LNCS*, pages 214–230, 2017.

[25] Clark W. Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanović, Tim King, Andrew Reynolds, and Cesare Tinelli. CVC4. In *23rd International Conference on Computer Aided Verification (CAV)*, volume 6806 of *LNCS*, pages 171–177. Springer, 2011.

[26] Clark W. Barrett, Pascal Fontaine, and Cesare Tinelli. The SMT-LIB Standard: Version 2.6. Technical report, Department of Computer Science, The University of Iowa, 2017. Available at `www.SMT-LIB.org`.

[27] Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. Satisfiability modulo theories. In Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 825–885. IOS Press, 2009.

[28] Michael A. Bekos, Michael Kaufmann, Stephen G. Kobourov, and Sankar Veeramoni. A note on maximum differential coloring of planar graphs. *J. Discrete Algorithms*, 29:1–7, 2014.

[29] Nicolas Beldiceanu and Evelyne Contejean. Introducing global constraints in CHIP. *Mathematical and Computer Modelling*, 20:97–123, 03 1996.

[30] Marco Benedetti and Hratch Mangassarian. QBF-Based Formal Verification: Experience and Perspectives. *JSAT*, 2008.

[31] Jeremias Berg, Matti Järvisalo, and Brandon Malone. Learning optimal bounded treewidth bayesian networks via maximum satisfiability. In *17th International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 86–95, 2014.

[32] David Bergman, André A. Ciré, Willem-Jan van Hoeve, and John N. Hooker. *Decision Diagrams for Optimization*. Artificial Intelligence: Foundations, Theory, and Algorithms. Springer, 2016.

[33] Christian Bessiere, Emmanuel Hebrard, Brahim Hnich, Zeynep Kiziltan, and Toby Walsh. SLIDE: A useful special case of the CARDPATH constraint. In *18th European Conference on Artificial Intelligence (ECAI)*, volume 178 of *Frontiers in Artificial Intelligence and Applications*, pages 475–479. IOS Press, 2008.

[34] Olaf Beyersdorff, Joshua Blinkhorn, and Meena Mahajan. Building strategies into QBF proofs. In *36th International Symposium on Theoretical Aspects of Computer Science, (STACS)*, volume 126 of *LIPIcs*, pages 14:1–14:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[35] Armin Biere. Resolve and expand. In *Selected Revised Papers of 7th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 3542 of *Lecture Notes in Computer Science*, pages 59–70. Springer, 2004.

[36] Armin Biere. Yet another local search solver and Lingeling and friends entering the SAT competition 2014. In Adrian Balint, Anton Belov, Marijn J. H. Heule, and Matti Järvisalo, editors, *Proc. of SAT Competition 2014 – Solver and Benchmark Descriptions*, Department of Computer Science Series of Publications B, pages 39–40. University of Helsinki, 2014.

[37] Armin Biere. CaDiCaL, Lingeling, Plingeling, Treengeling and YalSAT Entering the SAT Competition 2018. In Marijn J. H. Heule, Matti Järvisalo, and Martin Suda, editors, *Proc. of SAT Competition 2018 – Solver and Benchmark Descriptions*, volume B-2018-1 of *Department of Computer Science Series of Publications B*, pages 13–14. University of Helsinki, 2018.

[38] Armin Biere. CaDiCaL at the SAT Race 2019. In Marijn J. H. Heule, Matti Järvisalo, and Martin Suda, editors, *Proc. of SAT Race 2019 – Solver and Benchmark Descriptions*, volume B-2019-1 of *Department of Computer Science Series of Publications B*, pages 8–9. University of Helsinki, 2019.

[39] Armin Biere, Daniel Le Berre, Emmanuel Lonca, and Norbert Manthey. Detecting cardinality constraints in CNF. In *17th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 8561 of *LNCS*, pages 285–301. Springer, 2014.

[40] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic model checking without BDDs. In *5th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, volume 1579 of *LNCS*, pages 193–207. Springer, 1999.

[41] Armin Biere, Keijo Heljanko, and Siert Wieringa. AIGER 1.9 and beyond. Technical report, Institute for Formal Models and Verification, Johannes Kepler University, Altenbergerstr. 69, 4040 Linz, Austria, 2011.

[42] Armin Biere and Marijn J. H. Heule. The effect of scrambling CNFs. In Daniel Le Berre and Matti Järvisalo, editors, *Pragmatics of SAT*, volume 59 of *EPiC Series in Computing*, pages 111–126. EasyChair, 2018.

[43] Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[44] Armin Biere, Tom van Dijk, and Keijo Heljanko. Hardware model checking competition 2017. In *17th International Conference on Formal Methods in Computer-Aided Design (FMCAD)*, page 9. IEEE, 2017.

[45] Nikolaj Bjørner, Anh-Dung Phan, and Lars Fleckenstein. νZ - an optimizing SMT solver. In *21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 9035 of *LNCS*, pages 194–199. Springer, 2015.

[46] Jasmin Christian Blanchette, Mathias Fleury, Peter Lammich, and Christoph Weidenbach. A verified SAT solver framework with learn, forget, restart, and incrementality. *J. Autom. Reasoning*, 61(1-4):333–365, 2018.

[47] Jasmin Christian Blanchette, Mathias Fleury, and Christoph Weidenbach. A verified SAT solver framework with learn, forget, restart, and incrementality. In *8th International Joint Conference on Automated Reasoning (IJCAR)*, volume 9706 of *LNCS*, pages 25–44. Springer, 2016.

[48] Roderick Bloem, Nicolas Braud-Santoni, Vedad Hadžić, Uwe Egly, Florian Lonsing, and Martina Seidl. Expansion-based QBF solving without recursion. In *18th International Conference on Formal Methods in Computer-Aided Design (FMCAD)*, pages 1–10. IEEE, 2018.

[49] Miquel Bofill, Jordi Coll, Josep Suy, and Mateu Villaret. SAT encodings of pseudo-Boolean constraints with at-most-one relations. In *16th International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR)*, volume 11494 of *LNCS*, pages 112–128. Springer, 2019.

[50] Miquel Bofill, Víctor Muñoz, and Javier Murillo. Solving the wastewater treatment plant problem with SMT. *CoRR*, abs/1609.05367, 2016.

[51] Natashia Boland, Thomas Kalinowski, Hamish Waterer, and Lanbo Zheng. Mixed integer programming based maintenance scheduling for the hunter valley coal chain. *J. Scheduling*, 16(6):649–659, 2013.

[52] Aaron R. Bradley. SAT-based model checking without unrolling. In *12th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI)*, volume 6538 of *LNCS*, pages 70–87. Springer, 2011.

[53] Aaron R. Bradley and Zohar Manna. *The calculus of computation - decision procedures with applications to verification.* Springer, 2007.

[54] Aaron R. Bradley, Zohar Manna, and Henny B. Sipma. What's decidable about arrays? In *7th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI)*, volume 3855 of *LNCS*, pages 427–442. Springer, 2006.

[55] Ronen I. Brafman. A simplifier for propositional formulas with many binary clauses. In Bernhard Nebel, editor, *17th International Joint Conference on Artificial Intelligence, (IJCAI)*, pages 515–522. Morgan Kaufmann, 2001.

[56] Martin Brain, Vijay D'Silva, Leopold Haller, Alberto Griggio, and Daniel Kroening. An abstract interpretation of DPLL(T). In *14th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI)*, volume 7737 of *LNCS*, pages 455–475. Springer, 2013.

[57] Sebastian Brand, Nina Narodytska, Claude-Guy Quimper, Peter J. Stuckey, and Toby Walsh. Encodings of the SEQUENCE constraint. In *13th International Conference on Principles and Practice of Constraint Programming (CP)*, volume 4741 of *LNCS*, pages 210–224. Springer, 2007.

[58] Rémi Brochenin and Marco Maratea. Abstract solvers for quantified Boolean formulas and their applications. In *14th International Conference of the Italian Association for Artificial Intelligence (AI*IA)*, volume 9336 of *LNCS*, pages 205–217. Springer, 2015.

[59] Robert Brummayer and Armin Biere. Local two-level And-Inverter graph minimization without blowup. In *2nd Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS)*, 2006.

[60] Randal E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986.

[61] Randal E. Bryant. Binary decision diagrams. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 191–217. Springer, 2018.

[62] Randal E. Bryant and Miroslav N. Velev. Boolean satisfiability with transitivity constraints. *ACM Trans. Comput. Log.*, 3(4):604–627, 2002.

[63] Jerry R. Burch and David L. Dill. Automatic verification of pipelined microprocessor control. In *6th International Conference on Computer Aided Verification (CAV)*, volume 818 of *LNCS*, pages 68–80. Springer, 1994.

[64] Edmund K. Burke, Patrick De Causmaecker, Greet Vanden Berghe, and Hendrik Van Landeghem. The state of the art of nurse rostering. *J. Scheduling*, 7(6):441–499, 2004.

[65] Marco Cadoli, Marco Schaerf, Andrea Giovanardi, and Massimo Giovanardi. An algorithm to evaluate quantified Boolean formulae and its experimental evaluation. *J. Autom. Reasoning*, 28(2):101–142, 2002.

[66] Cristiano Calcagno, Dino Distefano, Jérémy Dubreil, Dominik Gabi, Pieter Hooimeijer, Martino Luca, Peter W. O'Hearn, Irene Papakonstantinou, Jim Purbrick, and Dulma Rodriguez. Moving fast with software verification. In *7th International Symposium on NASA Formal Methods (NFM)*, volume 9058 of *LNCS*, pages 3–11. Springer, 2015.

[67] Paola Cappanera. A survey on obnoxious facility location problems, 1999.

[68] Claudio Castellini, Enrico Giunchiglia, and Armando Tacchella. SAT-based planning in complex domains: Concurrency, constraints and nondeterminism. *Artif. Intell.*, 147(1-2):85–117, 2003.

[69] Karthekeyan Chandrasekaran, Richard M. Karp, Erick Moreno-Centeno, and Santosh S. Vempala. Algorithms for implicit hitting set problems. In *22nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 614–629. SIAM, 2011.

[70] Jingchao Chen. A new SAT encoding of the at-most-one constraint. *Proc. Constraint Modelling and Reformulation*, 2010.

[71] Yibin Chen, Sean Safarpour, João Marques-Silva, and Andreas G. Veneris. Automated design debugging with maximum satisfiability. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 29(11):1804–1817, 2010.

[72] Alonzo Church. A note on the entscheidungsproblem. *J. Symb. Log.*, 1(1):40–41, 1936.

[73] Alessandro Cimatti, Alberto Griggio, Bastiaan Joost Schaafsma, and Roberto Sebastiani. The MathSAT5 SMT solver. In *19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 7795 of *LNCS*, pages 93–107. Springer, 2013.

[74] Alessandro Cimatti, Alberto Griggio, Bastiaan Joost Schaafsma, and Roberto Sebastiani. A modular approach to MaxSAT modulo theories. In *SAT*, volume 7962 of *LNCS*. Springer, 2013.

[75] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement for symbolic model checking. *J. ACM*, 50(5):752–794, 2003.

[76] Stephen A. Cook. The complexity of theorem-proving procedures. In *3rd Annual ACM Symposium on Theory of Computing*, pages 151–158. ACM, 1971.

[77] William Craig. Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory. *J. Symb. Log.*, 22(3):269–285, 1957.

[78] Neil T. Dantam, Zachary K. Kingston, Swarat Chaudhuri, and Lydia E. Kavraki. An incremental constraint-based framework for task and motion planning. *I. J. Robotics Res.*, 37(10), 2018.

[79] Jessica Davies and Fahiem Bacchus. Solving MaxSAT by solving a sequence of simpler SAT instances. In *17th International Conference on Principles and Practice of Constraint Programming (CP)*, volume 6876 of *LNCS*, pages 225–239. Springer, 2011.

[80] Jessica Davies and Fahiem Bacchus. Postponing optimization to speed up MAXSAT solving. In *19th International Conference on Principles and Practice of Constraint Programming (CP)*, pages 247–262, 2013.

[81] Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: an efficient SMT solver. In *14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 4963 of *LNCS*, pages 337–340. Springer, 2008.

[82] Leonardo Mendonça de Moura, Harald Rueß, and Maria Sorea. Lemmas on demand for satisfiability solvers. *5th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, 2:244–251, 2002.

[83] Mehmet Dincbas, Helmut Simonis, and Pascal Van Hentenryck. Solving the car-sequencing problem in constraint logic programming. In *8th European Conference on Artificial Intelligence (ECAI)*, pages 290–295. Pitmann Publishing, London, 1988.

[84] Stefan Dobrev, Rastislav Královic, Dana Pardubská, L'ubomír Török, and Imrich Vrt'o. Antibandwidth and cyclic antibandwidth of hamming graphs. *Discret. Appl. Math.*, 161(10-11):1402–1408, 2013.

[85] Abraham Duarte, Rafael Martí, Mauricio G. C. Resende, and Ricardo M. A. Silva. GRASP with path relinking heuristics for the antibandwidth problem. *Networks*, 58(3):171–189, 2011.

[86] Niklas Eén and Armin Biere. Effective preprocessing in SAT through variable and clause elimination. In Fahiem Bacchus and Toby Walsh, editors,

8th International Conference on Theory and Applications of Satisfiability Testing (SAT), volume 3569 of *LNCS*, pages 61–75. Springer, 2005.

[87] Niklas Eén, Alan Mishchenko, and Nina Amla. A single-instance incremental SAT formulation of proof- and counterexample-based abstraction. In *10th International Conference on Formal Methods in Computer-Aided Design (FMCAD)*, pages 181–188. IEEE, 2010.

[88] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In *6th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 2919 of *LNCS*, pages 502–518. Springer, 2003.

[89] Niklas Eén and Niklas Sörensson. Temporal induction by incremental SAT solving. *Electr. Notes Theor. Comput. Sci.*, 89(4):543–560, 2003.

[90] Niklas Eén and Niklas Sörensson. Translating pseudo-Boolean constraints into SAT. *JSAT*, 2(1-4):1–26, 2006.

[91] Uwe Egly. On sequent systems and resolution for QBFs. In *15th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 7317 of *LNCS*, pages 100–113. Springer, 2012.

[92] Andreas T. Ernst, Houyuan Jiang, Mohan Krishnamoorthy, and David Sier. Staff scheduling and rostering: A review of applications, methods and models. *European Journal of Operational Research*, 153(1):3–27, 2004.

[93] Katalin Fazekas, Fahiem Bacchus, and Armin Biere. Implicit Hitting Set Algorithms for Maximum Satisfiability Modulo Theories. In Didier Galmiche, Stephan Schulz, and Roberto Sebastiani, editors, *9th International Joint Conference on Automated Reasoning (IJCAR)*, volume 10900 of *LNCS*, pages 134–151. Springer, 2018.

[94] Katalin Fazekas, Armin Biere, and Christoph Scholl. Incremental Inprocessing in SAT Solving. In Mikoláš Janota and Inês Lynce, editors, *22nd International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 11628 of *LNCS*, pages 136–154. Springer, 2019.

[95] Katalin Fazekas, Marijn J. H. Heule, Martina Seidl, and Armin Biere. Skolem Function Continuation for Quantified Boolean Formulas. In Sebastian Gabmeyer and Einar Broch Johnsen, editors, *11th International Conference on Tests and Proofs (TAP)*, volume 10375 of *Lecture Notes in Computer Science*, pages 129–138. Springer, 2017.

[96] Katalin Fazekas, Martina Seidl, and Armin Biere. A Duality-Aware Calculus for Quantified Boolean Formulas. In James H. Davenport, Viorel Negru, Tetsuo Ida, Tudor Jebelean, Dana Petcu, Stephen M. Watt, and Daniela Zaharie, editors, *18th International Symposium on Symbolic and*

Numeric Algorithms for Scientific Computing, (SYNASC), pages 181–186. IEEE, 2016.

[97] Katalin Fazekas, Markus Sinnl, Armin Biere, and Sophie Parragh. Duplex Encoding of Staircase At-Most-One Constraints for the Antibandwidth Problem. In Emmanuel Hebrard and Nysret Musliu, editors, *17th International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR)*, To be published. Springer, 2020.

[98] Thibaut Feydy and Peter J. Stuckey. Lazy clause generation reengineered. In *15th International Conference on Principles and Practice of Constraint Programming (CP)*, volume 5732 of *LNCS*, pages 352–366. Springer, 2009.

[99] J. A. Gallian. A dynamic survey of graph labeling. *The Electronic Journal of Combinatorics*, 16(6):1–219, 2009.

[100] Emden R. Gansner, Yifan Hu, and Stephen G. Kobourov. Gmap: Visualizing graphs and clusters as maps. In *IEEE Pacific Visualization Symposium PacificVis*, pages 201–208. IEEE Computer Society, 2010.

[101] Yeting Ge, Clark W. Barrett, and Cesare Tinelli. Solving quantified verification conditions using satisfiability modulo theories. *Ann. Math. Artif. Intell.*, 55(1-2):101–122, 2009.

[102] Yeting Ge and Leonardo Mendonça de Moura. Complete instantiation for quantified formulas in satisfiabiliby modulo theories. In *21st International Conference on Computer Aided Verification (CAV)*, volume 5643 of *LNCS*, pages 306–320. Springer, 2009.

[103] Allen Van Gelder. Contributions to the theory of practical quantified Boolean formula solving. In *18th International Conference on Principles and Practice of Constraint Programming (CP)*, volume 7514 of *LNCS*, pages 647–663. Springer, 2012.

[104] Ian P. Gent. Arc consistency in SAT. In *15th Eureopean Conference on Artificial Intelligence, (ECAI)*, pages 121–125. IOS Press, 2002.

[105] Enrico Giunchiglia, Paolo Marin, and Massimo Narizzano. Reasoning with quantified Boolean formulas. In Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 761–780. IOS Press, 2009.

[106] Enrico Giunchiglia, Paolo Marin, and Massimo Narizzano. QuBE 7.0. *J. Satisf. Boolean Model. Comput.*, 7(2-3):83–88, 2010.

[107] Enrico Giunchiglia, Massimo Narizzano, and Armando Tacchella. Quantified Boolean formulas satisfiability library (qbflib). *Website:* `http://www.qbflib.org`, 2001.

[108] Enrico Giunchiglia, Massimo Narizzano, and Armando Tacchella. Clause/Term resolution and learning in the evaluation of quantified Boolean formulas. *J. Artif. Intell. Res.*, 26:371–416, 2006.

[109] Stephan Gocht and Tomáš Balyo. Accelerating SAT based planning with incremental SAT solving. In *27th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 135–139. AAAI Press, 2017.

[110] Patrice Godefroid, Michael Y. Levin, and David A. Molnar. SAGE: whitebox fuzzing for security testing. *Commun. ACM*, 55(3):40–44, 2012.

[111] Alexandra Goultiaeva and Fahiem Bacchus. Off the Trail: Re-examining the CDCL Algorithm. In *SAT*, 2012.

[112] Alexandra Goultiaeva, Allen Van Gelder, and Fahiem Bacchus. A uniform approach for generating proofs and strategies for both true and false QBF formulas. In *22nd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 546–553. IJCAI/AAAI, 2011.

[113] Alexandra Goultiaeva, Martina Seidl, and Armin Biere. Bridging the gap between dual propagation and CNF-based QBF solving. In *Design, Automation and Test in Europe (DATE)*, pages 811–814. EDA Consortium San Jose, CA, USA / ACM DL, 2013.

[114] Liana Hadarean, Kshitij Bansal, Dejan Jovanović, Clark W. Barrett, and Cesare Tinelli. A tale of two solvers: Eager and lazy approaches to bit-vectors. In *26th International Conference on Computer Aided Verification (CAV)*, volume 8559 of *LNCS*, pages 680–695. Springer, 2014.

[115] William K. Hale. Frequency assignment: Theory and applications. *Proceedings of the IEEE*, 68(12):1497–1514, 1980.

[116] Hyojung Han and Fabio Somenzi. Alembic: An efficient algorithm for CNF preprocessing. In *44th Design Automation Conference (DAC)*, pages 582–587. IEEE, 2007.

[117] Mostafa Hassan, Caterina Urban, Marco Eilers, and Peter Müller. MaxSMT-Based type inference for Python 3. In *30th International Conference on Computer Aided Verification (CAV)*, volume 10982 of *LNCS*, pages 12–19. Springer, 2018.

[118] Marijn J. H. Heule, Matti Järvisalo, and Armin Biere. Efficient CNF simplification based on binary implication graphs. In *14th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 6695 of *LNCS*, pages 201–215. Springer, 2011.

[119] Marijn J. H. Heule, Matti Järvisalo, and Armin Biere. Revisiting hyper binary resolution. In *10th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR)*, volume 7874 of *LNCS*, pages 77–93. Springer, 2013.

[120] Marijn J. H. Heule, Matti Järvisalo, Florian Lonsing, Martina Seidl, and Armin Biere. Clause elimination for SAT and QSAT. *JAIR*, 2015.

[121] Marijn J. H. Heule, Matti Järvisalo, and Martin Suda, editors. *Proc. of SAT Competition 2018 – Solver and Benchmark Descriptions*, volume B-2018-1 of *Department of Computer Science Series of Publications B*. University of Helsinki, 2018.

[122] Marijn J. H. Heule, Matti Järvisalo, and Martin Suda, editors. *Proc. of SAT Race 2019 – Solver and Benchmark Descriptions*, volume B-2019-1 of *Department of Computer Science Series*. University of Helsinki, 2019.

[123] Marijn J. H. Heule, Warren A. Hunt Jr., and Nathan Wetzler. Verifying refutations with extended resolution. In *24th International Conference on Automated Deduction (CADE)*, volume 7898 of *LNCS*, pages 345–359. Springer, 2013.

[124] Marijn J. H. Heule, Benjamin Kiesl, and Armin Biere. Short proofs without new variables. In *26th International Conference on Automated Deduction (CADE)*, volume 10395 of *LNCS*, pages 130–147. Springer, 2017.

[125] Marijn J. H. Heule, Benjamin Kiesl, and Armin Biere. Strong extension-free proof systems. *J. Autom. Reasoning*, 64(3):533–554, 2020.

[126] Randy Hickey and Fahiem Bacchus. Speeding up assumption-based SAT. In *22nd International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 11628 of *LNCS*, pages 164–182. Springer, 2019.

[127] Georg Hofferek. *Controller Synthesis with Uninterpreted Functions*. PhD thesis, Graz University of Technology, 2014.

[128] Georg Hofferek and Roderick Bloem. Controller synthesis for pipelined circuits using uninterpreted functions. In *9th IEEE/ACM International Conference on Formal Methods and Models for Codesign (MEMOCODE)*, pages 31–42. IEEE, 2011.

[129] Georg Hofferek, Ashutosh Gupta, Bettina Könighofer, Jie-Hong Roland Jiang, and Roderick Bloem. Synthesizing multiple Boolean functions using interpolation on a single proof. In *13th International Conference on Formal Methods in Computer-Aided Design (FMCAD)*, pages 77–84. IEEE, 2013.

[130] Steffen Hölldobler and Van-Hau Nguyen. On SAT-Encodings of the at-most-one constraint. In George Katsirelos and Claude-Guy Quimper, editors, *12th International Workshop on Constraint Modelling and Reformulation*, pages 1–17, 2013.

[131] John N. Hooker. Solving the incremental satisfiability problem. *J. Log. Program.*, 15(1&2):177–186, 1993.

[132] Mikoláš Janota. *SAT Solving in Interactive Configuration*. PhD thesis, University College Dublin, November 2010.

[133] Mikoláš Janota. On Q-resolution and CDCL QBF solving. In *19th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 9710 of *LNCS*, pages 402–418. Springer, 2016.

[134] Mikoláš Janota, William Klieber, João Marques-Silva, and Edmund M. Clarke. Solving QBF with counterexample guided refinement. *Artif. Intell.*, 2016.

[135] Mikoláš Janota and João Marques-Silva. Expansion-based QBF solving versus Q-resolution. *Theor. Comput. Sci.*, 577, 2015.

[136] Matti Järvisalo and Armin Biere. Reconstructing solutions after blocked clause elimination. In *13th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 6175 of *LNCS*, pages 340–345. Springer, 2010.

[137] Matti Järvisalo, Armin Biere, and Marijn J. H. Heule. Blocked clause elimination. In *16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 6015 of *LNCS*, pages 129–144. Springer, 2010.

[138] Matti Järvisalo, Marijn J. H. Heule, and Armin Biere. Inprocessing rules. In *6th International Joint Conference on Automated Reasoning (IJCAR)*, volume 7364 of *LNCS*, pages 355–370. Springer, 2012.

[139] Jie-Hong Roland Jiang, Hsuan-Po Lin, and Wei-Lun Hung. Interpolating functions from large Boolean relations. In *International Conference on Computer-Aided Design (ICCAD)*, pages 779–784, 2009.

[140] Martin Jonáš and Jan Strejček. Solving quantified bit-vector formulas using binary decision diagrams. In *19th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 9710 of *LNCS*, pages 267–283. Springer, 2016.

[141] Martin Jonáš and Jan Strejček. Is satisfiability of quantified bit-vector formulas stable under bit-width changes? (experimental paper). In *22nd International Conference on Logic for Programming, Artificial Intelligence*

and Reasoning (LPAR), volume 57 of *EPiC Series in Computing*, pages 488–497. EasyChair, 2018.

[142] Toni Jussila and Armin Biere. Compressing BMC encodings with QBF. *Electron. Notes Theor. Comput. Sci.*, 174(3):45–56, 2007.

[143] Toni Jussila, Armin Biere, Carsten Sinz, Daniel Kröning, and Christoph M. Wintersteiger. A first step towards a unified proof checker for QBF. In *10th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 4501 of *LNCS*, pages 201–214. Springer, 2007.

[144] Henry A. Kautz and Bart Selman. Planning as satisfiability. In *10th European Conference on Artificial Intelligence (ECAI)*, pages 359–363. John Wiley and Sons, 1992.

[145] Hans Kleine Büning, Marek Karpinski, and Andreas Flögel. Resolution for quantified Boolean formulas. *Inf. Comput.*, 117(1):12–18, 1995.

[146] William Klieber, Samir Sapra, Sicun Gao, and Edmund M. Clarke. A non-prenex, non-clausal QBF solver with game-state learning. In *13th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 6175 of *LNCS*, pages 128–142. Springer, 2010.

[147] Donald E. Knuth. *The Art of Computer Programming, Volume 4B, Fascicle 6: Satisfiability.* Addison-Wesley, 2015.

[148] Laura Kovács and Andrei Voronkov. First-order theorem proving and vampire. In *25th International Conference on Computer Aided Verification (CAV)*, volume 8044 of *LNCS*, pages 1–35. Springer, 2013.

[149] Jan Krajíček and Pavel Pudlák. Quantified propositional calculi and fragments of bounded arithmetic. *Math. Log. Q.*, 36(1):29–46, 1990.

[150] Daniel Kroening and Ofer Strichman. A framework for satisfiability modulo theories. *Formal Asp. Comput.*, 21(5):485–494, 2009.

[151] Daniel Kroening and Ofer Strichman. *Decision Procedures - An Algorithmic Point of View, Second Edition.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 2016.

[152] Andreas Kuehlmann, Viresh Paruthi, Florian Krohm, and Malay K. Ganai. Robust Boolean reasoning for equivalence checking and functional property verification. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 21(12):1377–1394, 2002.

[153] Oliver Kullmann. On a generalization of extended resolution. *Discrete Applied Mathematics*, 96-97:149–176, 1999.

[154] Stefan Kupferschmid, Matthew D. T. Lewis, Tobias Schubert, and Bernd Becker. Incremental preprocessing methods for use in BMC. *Formal Methods in System Design*, 39(2):185–204, 2011.

[155] Jean-Marie Lagniez and Armin Biere. Factoring out assumptions to speed up MUS extraction. In *16th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 7962 of *LNCS*, pages 276–292. Springer, 2013.

[156] Tracy Larrabee. Test pattern generation using Boolean satisfiability. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 11(1):4–15, 1992.

[157] Reinhold Letz. Lemma and model caching in decision procedures for quantified Boolean formulas. In *International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX)*, volume 2381 of *LNCS*, pages 160–175. Springer, 2002.

[158] Joseph Y.-T. Leung, Oliver Vornberger, and James D. Witthoff. On some variants of the bandwidth minimization problem. *SIAM J. Comput.*, 13(3):650–667, 1984.

[159] Yi Li, Aws Albarghouthi, Zachary Kincaid, Arie Gurfinkel, and Marsha Chechik. Symbolic optimization with SMT solvers. In *POPL*, pages 607–618. ACM, 2014.

[160] Mark H. Liffiton and Jordyn C. Maglalang. A cardinality solver: More expressive constraints for free - (poster presentation). In *15th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 7317 of *LNCS*, pages 485–486. Springer, 2012.

[161] Florian Lonsing and Armin Biere. Nenofex: Expanding NNF for QBF solving. In *11th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 4996 of *LNCS*, pages 196–210. Springer, 2008.

[162] Florian Lonsing and Armin Biere. DepQBF: A Dependency-Aware QBF Solver. *JSAT*, 7, 2010.

[163] Florian Lonsing and Uwe Egly. Incremental QBF solving. In *20th International Conference on Principles and Practice of Constraint Programming (CP)*, volume 8656 of *LNCS*, pages 514–530. Springer, 2014.

[164] Florian Lonsing and Uwe Egly. Incremental QBF solving by DepQBF. In *4th International Congress on Mathematical Software (ICMS)*, volume 8592 of *LNCS*, pages 307–314. Springer, 2014.

[165] Florian Lonsing and Uwe Egly. Evaluating QBF solvers: Quantifier alternations matter. In *24th International Conference on Principles and*

Practice of Constraint Programming (CP), volume 11008 of *LNCS*, pages 276–294. Springer, 2018.

[166] Florian Lonsing, Uwe Egly, and Martina Seidl. Q-Resolution with Generalized Axioms. In *19th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 9710 of *LNCS*, pages 435–452. Springer, 2016.

[167] Florian Lonsing, Martina Seidl, and Allen Van Gelder. The QBF gallery: Behind the scenes. *Artif. Intell.*, 237:92–114, 2016.

[168] Manuel Lozano, Abraham Duarte, Francisco Gortázar, and Rafael Martí. Variable neighborhood search with ejection chains for the antibandwidth problem. *J. Heuristics*, 18(6):919–938, 2012.

[169] Mao Luo, Chu-Min Li, Fan Xiao, Felip Manyà, and Zhipeng Lü. An effective learnt clause minimization approach for CDCL SAT solvers. In *26th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 703–711. ijcai.org, 2017.

[170] Sharad Malik and Lintao Zhang. Boolean satisfiability from theoretical hardness to practical success. *Commun. ACM*, 52(8):76–82, 2009.

[171] Panagiotis Manolios, Jorge Pais, and Vasilis Papavasileiou. The Inez mathematical programming modulo theories framework. In *27th International Conference on Computer Aided Verification (CAV)*, pages 53–69, 2015.

[172] Norbert Manthey. Riss 7. In Tomáš Balyo, Marijn J. H. Heule, and Matti Järvisalo, editors, *Proc. of SAT Competition 2017 – Solver and Benchmark Descriptions*, volume B-2017-1 of *Department of Computer Science Series of Publications B*, page 29. University of Helsinki, 2017.

[173] Norbert Manthey, Marijn J. H. Heule, and Armin Biere. Automated reencoding of Boolean formulas. In *Revised Selected Papers of the 8th International Haifa Verification Conference on Hardware and Software: Verification and Testing (HVC)*, volume 7857 of *LNCS*, pages 102–117. Springer, 2012.

[174] Christos T. Maravelias. On the combinatorial structure of discrete-time MIP formulations for chemical production scheduling. *Computers & Chemical Engineering*, 38:204–212, 2012.

[175] João Marques-Silva, Josep Argelich, Ana Graça, and Inês Lynce. Boolean lexicographic optimization: algorithms & applications. *Ann. Math. AI*, 62(3-4):317–343, 2011.

[176] João Marques-Silva, Inês Lynce, and Sharad Malik. Conflict-driven clause learning SAT solvers. In Armin Biere, Marijn J. H. 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.

[177] Ruben Martins, Saurabh Joshi, Vasco M. Manquinho, and Inês Lynce. Incremental cardinality constraints for MaxSAT. In *20th International Conference on Principles and Practice of Constraint Programming (CP)*, volume 8656 of *LNCS*, pages 531–548. Springer, 2014.

[178] Ruben Martins, Saurabh Joshi, Vasco M. Manquinho, and Inês Lynce. On using incremental encodings in unsatisfiability-based MaxSAT solving. *JSAT*, 9:59–81, 2014.

[179] Orly Meir and Ofer Strichman. Yet another decision procedure for equality logic. In *17th International Conference on Computer Aided Verification (CAV)*, volume 3576 of *LNCS*, pages 307–320. Springer, 2005.

[180] Zevi Miller and Dan Pritikin. On the separation number of a graph. *Networks*, 19(6):651–666, 1989.

[181] Erick Moreno-Centeno and Richard M. Karp. The implicit hitting set approach to solve combinatorial optimization problems with an application to multigenome alignment. *Oper. Res.*, 61(2):453–468, 2013.

[182] António Morgado, Federico Heras, Mark H. Liffiton, Jordi Planes, and João Marques-Silva. Iterative and core-guided MaxSAT solving: A survey and assessment. *Constraints*, 18(4):478–534, 2013.

[183] Alexander Nadel. Boosting minimal unsatisfiable core extraction. In *10th International Conference on Formal Methods in Computer-Aided Design (FMCAD)*, pages 221–229. IEEE, 2010.

[184] Alexander Nadel and Vadim Ryvchin. Efficient SAT solving under assumptions. In *15th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 7317 of *LNCS*, pages 242–255. Springer, 2012.

[185] Alexander Nadel, Vadim Ryvchin, and Ofer Strichman. Preprocessing in incremental SAT. In *15th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 7317 of *LNCS*, pages 256–269. Springer, 2012.

[186] Alexander Nadel, Vadim Ryvchin, and Ofer Strichman. Ultimately incremental SAT. In *17th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 8561 of *LNCS*, pages 206–218. Springer, 2014.

[187] Nina Narodytska and Fahiem Bacchus. Maximum satisfiability using core-guided MaxSAT resolution. In *28th AAAI Conference on Artificial Intelligence (AAAI)*, pages 2717–2723. AAAI Press, 2014.

[188] Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. MiniZinc: Towards a standard CP modelling language. In Christian Bessiere, editor, *13th International Conference on Principles and Practice of Constraint Programming (CP)*, volume 4741 of *LNCS*, pages 529–543. Springer, 2007.

[189] Van-Hau Nguyen. SAT encodings of finite-CSP domains: A survey. In *8th International Symposium on Information and Communication Technology (SOICT)*, pages 84–91. ACM, 2017.

[190] Aina Niemetz, Mathias Preiner, Andrew Reynolds, Clark W. Barrett, and Cesare Tinelli. Solving quantified bit-vectors using invertibility conditions. In *30th International Conference on Computer Aided Verification (CAV)*, volume 10982 of *LNCS*, pages 236–255. Springer, 2018.

[191] Aina Niemetz, Mathias Preiner, Clifford Wolf, and Armin Biere. Btor2, BtorMC and Boolector 3.0. In *30th International Conference on Computer Aided Verification (CAV)*, volume 10981 of *LNCS*, pages 587–595. Springer, 2018.

[192] Robert Nieuwenhuis and Albert Oliveras. On SAT modulo theories and optimization problems. In *9th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 4121 of *LNCS*, pages 156–169. Springer, 2006.

[193] Robert Nieuwenhuis and Albert Oliveras. Fast congruence closure and extensions. *Inf. Comput.*, 205(4):557–580, 2007.

[194] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving SAT and SAT Modulo Theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). *J. ACM*, 53(6):937–977, 2006.

[195] Christos H. Papadimitriou. *Computational complexity.* Addison-Wesley, 1994.

[196] Tobias Philipp and Peter Steinke. PBLib - A library for encoding pseudo-Boolean constraints into CNF. In *18th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 9340 of *LNCS*, pages 9–16. Springer, 2015.

[197] Cédric Piette, Youssef Hamadi, and Lakhdar Sais. Vivifying propositional clausal formulae. In *18th European Conference on Artificial Intelligence, (ECAI)*, volume 178 of *Frontiers in Artificial Intelligence and Applications*, pages 525–529. IOS Press, 2008.

[198] David A. Plaisted and Steven Greenbaum. A structure-preserving clause form translation. *J. Symb. Comput.*, 2(3):293–304, 1986.

[199] Amir Pnueli, Yoav Rodeh, Ofer Strichman, and Michael Siegel. Deciding equality formulas by small domains instantiations. In *11th International Conference on Computer Aided Verification (CAV)*, volume 1633 of *LNCS*, pages 455–469. Springer, 1999.

[200] Amir Pnueli and Ofer Strichman. Reduced functional consistency of uninterpreted functions. *Electron. Notes Theor. Comput. Sci.*, 144(2):53–65, 2006.

[201] Mukul R. Prasad, Armin Biere, and Aarti Gupta. A survey of recent advances in SAT-based formal verification. *Int. J. Softw. Tools Technol. Transf.*, 7(2):156–173, 2005.

[202] Mathias Preiner, Aina Niemetz, and Armin Biere. Better lemmas with lambda extraction. In *15th International Conference on Formal Methods in Computer-Aided Design (FMCAD)*, pages 128–135. IEEE, 2015.

[203] Steven David Prestwich. CNF encodings. In Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 75–97. IOS Press, 2009.

[204] Luca Pulina and Martina Seidl. The 2016 and 2017 QBF solvers evaluations (QBFEVAL'16 and QBFEVAL'17). *Artif. Intell.*, 274:224–248, 2019.

[205] André Raspaud, Heiko Schröder, Ondrej Sýkora, L'ubomír Török, and Imrich Vrt'o. Antibandwidth and cyclic antibandwidth of meshes and hypercubes. *Discret. Math.*, 309(11):3541–3552, 2009.

[206] Andrew Reynolds, Haniel Barbosa, and Pascal Fontaine. Revisiting enumerative instantiation. In *24th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 10806 of *LNCS*, pages 112–131. Springer, 2018.

[207] Jussi Rintanen. Improvements to the evaluation of quantified Boolean formulae. In Thomas Dean, editor, *16th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1192–1197. Morgan Kaufmann, 1999.

[208] Eduardo Rodriguez-Tello, Hillel Romero-Monsivais, José Ramírez-Torres, and Frédéric Lardeux. Harwell-Boeing graphs for the CB problem. https://www.researchgate.net/publication/272022702_Harwell-Boeing_graphs_for_the_CB_problem, 2015.

[209] Olivier Roussel and Vasco M. Manquinho. Pseudo-Boolean and cardinality constraints. In Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 695–733. IOS Press, 2009.

[210] Ashish Sabharwal, Carlos Ansótegui, Carla P. Gomes, Justin W. Hart, and Bart Selman. QBF modeling: Exploiting player symmetry for simplicity and efficiency. In *9th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 4121 of *LNCS*, pages 382–395. Springer, 2006.

[211] Paul Saikko. *Implicit Hitting Set Algorithms for Constraint Optimization*. PhD thesis, University of Helsinki, Helsinki, Finland, 2019.

[212] Paul Saikko, Johannes Peter Wallner, and Matti Järvisalo. Implicit hitting set algorithms for reasoning beyond NP. In *15th International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 104–113. AAAI Press, 2016.

[213] Edward J. Schwartz, Thanassis Avgerinos, and David Brumley. All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask). In *31st IEEE Symposium on Security and Privacy (S&P)*, pages 317–331. IEEE Computer Society, 2010.

[214] Jennifer A. Scott and Yifan Hu. Level-based heuristics and hill climbing for the antibandwidth maximization problem. *Numerical Lin. Alg. with Applic.*, 21(1):51–67, 2014.

[215] Roberto Sebastiani. Lazy satisability modulo theories. *J. Satisf. Boolean Model. Comput.*, 3(3-4):141–224, 2007.

[216] Roberto Sebastiani and Silvia Tomasi. Optimization modulo theories with linear rational costs. *ACM Trans. Comput. Log.*, 16(2):12:1–12:43, 2015.

[217] Roberto Sebastiani and Patrick Trentin. OptiMathSAT: A tool for optimization modulo theories. In *27th International Conference on Computer Aided Verification (CAV)*, pages 447–454, 2015.

[218] Roberto Sebastiani and Patrick Trentin. On optimization modulo theories, MaxSMT and sorting networks. In *23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 10206 of *LNCS*, pages 231–248, 2017.

[219] Ankit Shukla, Armin Biere, Luca Pulina, and Martina Seidl. A survey on applications of quantified Boolean formulas. In *31st IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 78–84. IEEE, 2019.

[220] Markus Sinnl. A note on computational approaches for the antibandwidth problem. *CoRR*, abs/1910.03367, 2019.

[221] Carsten Sinz. Towards an optimal CNF encoding of Boolean cardinality constraints. In *11th International Conference on Principles and Practice of Constraint Programming (CP)*, volume 3709 of *LNCS*, pages 827–831. Springer, 2005.

[222] Christine Solnon, Van-Dat Cung, Alain Nguyen, and Christian Artigues. The car sequencing problem: Overview of state-of-the-art methods and industrial case-study of the ROADEF'2005 challenge problem. *European Journal of Operational Research*, 191(3):912–927, 2008.

[223] Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending SAT solvers to cryptographic problems. In *12th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 5584 of *LNCS*, pages 244–257. Springer, 2009.

[224] Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time: Preliminary report. In *5th Annual ACM Symposium on Theory of Computing*, pages 1–9, 1973.

[225] Ofer Strichman. Pruning techniques for the SAT-based bounded model checking problem. In *Correct Hardware Design and Verification Methods, 11th IFIP WG 10.5 Advanced Research Working Conference (CHARME)*, volume 2144 of *LNCS*, pages 58–70. Springer, 2001.

[226] Peter J. Stuckey. Lazy clause generation: Combining the power of SAT and CP (and MIP?) solving. In *7th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR)*, volume 6140 of *LNCS*, pages 5–9. Springer, 2010.

[227] G. S. Tseitin. On the complexity of derivation in the propositional calculus. In A. O. Slisenko, editor, *Studies in Constr. Math. and Math. Logic, Part II.* , 1968.

[228] J. M. van den Akker. *LP-based solution methods for single-machine scheduling problems*. PhD thesis, Technische Universiteit Eindhoven - Department of Mathematics and Computer Science, 1994.

[229] Willem Jan van Hoeve, Gilles Pesant, Louis-Martin Rousseau, and Ashish Sabharwal. Revisiting the sequence constraint. In *12th International Conference on Principles and Practice of Constraint Programming (CP)*, volume 4204 of *LNCS*, pages 620–634. Springer, 2006.

[230] Juan Pablo Vielma. Mixed integer linear programming formulation techniques. *Siam Review*, 57(1):3–57, 2015.

[231] Yakir Vizel, Georg Weissenbacher, and Sharad Malik. Boolean satisfiability solvers and their applications in model checking. *Proceedings of the IEEE*, 103(11):2021–2035, 2015.

[232] Xiaohan Wang, Xiaolin Wu, and Sorina Dumitrescu. On explicit formulas for bandwidth and antibandwidth of hypercubes. *Discret. Appl. Math.*, 157(8):1947–1952, 2009.

[233] Christoph Weidenbach, Dilyana Dimova, Arnaud Fietzke, Rohit Kumar, Martin Suda, and Patrick Wischnewski. SPASS version 3.5. In *22nd International Conference on Automated Deduction (CADE)*, volume 5663 of *LNCS*, pages 140–145. Springer, 2009.

[234] Christoph M. Wintersteiger, Youssef Hamadi, and Leonardo Mendonça de Moura. Efficiently solving quantified bit-vector formulas. *Formal Methods in System Design*, 42(1):3–23, 2013.

[235] Lin Yixun and Yuan JinJiang. The dual bandwidth problem for graphs. *Journal of Zhengzhou University*, 35(1), 2003.

[236] Lei Zhang and Fahiem Bacchus. MaxSAT heuristics for cost optimal planning. In *26th AAAI Conference on Artificial Intelligence (AAAI)*, 2012.

[237] Lintao Zhang. Solving QBF by combining conjunctive and disjunctive normal forms. In *21st National Conference on Artificial Intelligence and the 18th Innovative Applications of Artificial Intelligence Conference*, pages 143–150. AAAI Press, 2006.

[238] Lintao Zhang and Sharad Malik. Conflict driven learning in a quantified Boolean satisfiability solver. In *IEEE/ACM International Conference on Computer-aided Design (ICCAD)*, pages 442–449. ACM / IEEE Computer Society, 2002.

[239] Lintao Zhang and Sharad Malik. Towards a symmetric treatment of satisfaction and conflicts in quantified Boolean formula evaluation. In *8th International Conference on Principles and Practice of Constraint Programming (CP)*, volume 2470 of *LNCS*, pages 200–215. Springer, 2002.