JOHANNES KEPLER
UNIVERSITY LINZ

Submitted by
Dipl.-Inform.
Sibylle Rosa Natalina Möhle-Rotondi

Submitted at
LIT Secure and Correct Systems Lab

Supervisor and
First Evaluator
Prof.
Dr. Armin Biere

Second Evaluator
Prof.
Adnan Darwiche, Ph.D.

June 2022

Formalizing Methods for Propositional Model Counting and Enumeration

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

Dreams don’t work unless you do.

— John C. Maxwell

Dedicated to Frank, Dario, and Silvan
who always supported my dreams.

ABSTRACT

Many real-world problems, such as probabilistic reasoning, can be formulated as the task of counting the models of a propositional formula, called #SAT. A model of a formula is an assignment to its variables such that the formula evaluates to true. Related to model counting is model enumeration, or All-SAT, in which the models of a formula are recorded. It is applied, e. g., in model checking and verification. Both #SAT and All-SAT are therefore of practical relevance.

In principle, we could check for every possible assignment whether it is a model of the formula. However, formulae stemming from real-world applications might be defined over thousands or millions of variables, and the number of possible assignments is exponential in the number of variables occurring in the formula. Clearly, checking them one by one is prohibitive. Therefore, state-of-the-art #SAT solvers partition the input formula into subformulae over pairwise disjoint sets of variables, called components, process them separately, and then combine the results. Each subformula is defined over a potentially small subset of the set of input variables, and the search space to be processed in each computation might be significantly smaller than the whole search space. Another means to reduce the number of assignments to be checked is to detect partial models, i. e., models in which not all variables occur. A partial model represents a set of total models whose size is exponential in the number of unassigned variables. The corresponding total assignments need not be checked one by one, hence partial model detection bears the potential to significantly reduce work in both space and time.

In this thesis, we develop and rigorously formalize methods for propositional model counting and enumeration with a focus on the detection of partial models. We first present two dual model counting methods. Simultaneously processing the formula and its negation enables us to detect short partial models. Our first approach is based on the Davis-Putnam-Loveland-Logemann (DPLL) algorithm. Our second method implements methods used in modern SAT solvers, such as conflict-driven clause learning (CDCL) and conflict-driven backjumping.

However, backjumping might cause the solver to repeat assignments just undone. To avoid these repetitions at least in part, conflict-driven clause learning was combined with chronological backtracking. In the context of model enumeration and counting, this allows for efficiently escaping search space regions without models while exploring the search space neighboring previously detected models. We formalize this method, which is called chronological CDCL, and its application to non-dual model counting and provide correctness proofs.

Partial models can be detected by checking whether a partial assignment already logically entails the input formula. We present this entailment check in four flavors of different strengths and computational complexity some of which make use of dual reasoning. Since entailment checks might be expensive, we finally present a method for shrinking total models by means of dual reasoning.

The focus of our work is on formalization and proofs. Preliminary results, either theoretical or experimental, show that the methods presented in this thesis enable us to find short partial models. Our methods are applicable in various domains and point out future directions for research in #SAT and All-SAT.

ZUSAMMENFASSUNG

Viele praktisch relevante Anwendungen wie das probabilistische Schliessen können als die Aufgabe formuliert werden, die Modelle einer aussagenlogischen Formel zu zählen. Diese Aufgabe wird #SAT genannt. Eine Zuweisung von Wahrheitswerten zu den Variablen einer Formel ist ein Modell einer aussagenlogischen Formel, wenn sie unter dieser Variablenbelegung zu wahr evaluiert. Eine ähnliche Aufgabenstellung ist All-SAT, das Aufzählen aller Modelle einer Formel. Anwendungsbeispiele wie Model Checking und Verifikation unterstreichen die Relevanz von #SAT und All-SAT in der Praxis.

Jede mögliche Wertzuweisung zu den Variablen einzeln zu testen ist bei Problemen aus der Praxis nicht durchführbar, da die Formeln Abertausende oder Millionen Variablen enthalten können. Moderne #SAT-Solver partitionieren die Formel daher in Teilformeln mit paarweise disjunkten Variablenmengen, verarbeiten diese einzeln und kombinieren die Resultate. Die Anzahl der in den einzelnen Berechnungen zu testenden Zuweisungen wird dadurch reduziert. Diese Reduktion wird auch durch das Finden von partiellen Modellen erreicht. In einem partiellen Modell kommen nicht alle Variablen vor, und es repräsentiert eine Menge von totalen Modellen, welche nicht einzeln getestet werden müssen, was eine signifikante Reduktion von Rechenzeit und Speicherbedarf zur Folge hat.

In dieser Dissertation entwickeln wir Methoden und Formalisierungen für das Zählen und Aufzählen von Modellen aussagenlogischer Formeln mit dem Schwerpunkt auf dem Auffinden von partiellen Modellen. Zuerst stellen wir zwei duale Methoden für das Zählen von Modellen vor. Die simultane Verarbeitung der Formel und ihrer Negation ermöglicht das Finden von kurzen partiellen Modellen. Unser erster Ansatz basiert auf dem Davis-Putnam-Loveland-Logemann-(DPLL)-Algorithmus. Unser zweites Verfahren unterstützt Methoden, welche in modernen SAT-Solvern verwendet werden, wie zum Beispiel das konfliktbasierte Lernen von Klauseln (CDCL) sowie das konfliktgetriebene Backtracking.

Nach dem Backtracking wiederholt der Solver oft Zuweisungen, die er soeben rückgängig gemacht hat. Um diese Wiederholungen zumindest teilweilse zu vermeiden, wurde CDCL mit chronologischem Backtracking kombiniert. Bezogen auf das Zählen von Modellen ermöglicht dies einerseits, Suchbereiche, welche kein Modell enthalten, effizient zu verlassen, und andererseits, Suchbereich in der Nachbarschaft von gefundenen Modellen zu untersuchen. Wir formalisieren chronologisches CDCL und seine Anwendung auf das nicht-duale Zählen von Modellen und liefern formale Korrektheitsbeweise.

Partielle Modelle sind auch partielle Zuweisungen, aus welchen die Formel gefolgert werden kann. Wir präsentieren den zugehörigen Test in vier Varianten unterschiedlicher Mächtigkeit und Rechenkomplexität, von welchen einige duales Schliessen verwenden. Da diese Tests aufwändig sein können, stellen wir zuletzt ein Verfahren für das Kürzen von totalen Modellen mittels dualen Schliessens vor.

Der Fokus in unserer Arbeit liegt auf Formalisierungen und Beweisen. Erste Resultate sowohl theoretischer als auch experimenteller Natur zeigen, dass die vorgestellten Verfahren das Finden von kurzen partiellen Modellen ermöglichen. Unsere Methoden sind für verschiedene Anwendungsbereiche geeignet und zeigen zukünftige Forschungsrichtungen in #SAT und All-SAT auf.

ACKNOWLEDGEMENTS

With this thesis, a chapter of my life is closed and another opened. Looking back I see many persons I would like to thank for supporting me and my dream to join academia for research.

My supervisor Armin Biere gave me the freedom I requested for my research and the guidance I needed. I want to thank him for his support, the valuable feedback he provided on my work, and for many fruitful discussions.

I am thankful to Christoph Beierle who arouse my interest in logic during my studies at the FernUniversität in Hagen and supported my wish to pursue an academic career.

Life does not follow our plans, and during my PhD I joined various institutes. I want to thank the heads of these groups giving me the opportunity to join them. They supported me and gave me the freedom I needed. I am thankful to my colleagues for many agreeable coffee breaks both with and without the usage of a whiteboard, for group meetings and undertakings. I want to thank Tobias Philipp and Christoph Wernhard for many interesting technical discussions at TU Dresden, where my journey started. Tobias, besides sharing office with me, arose my interest in formalization and guided my first steps in this challenging world. Christoph provided me with a view on logic from a different perspective, and his PIE environment was of great help in the implementation of my dual model counter in SWI-Prolog. I would like to thank Mathias Fleury for teaching me many implementation details of SAT solvers, for many fruitful discussions and for his valuable feedback on my work and of course for his company in conquering Upper Austrian summits. I am grateful to Martina Seidl who showed me the potential of my work in a different research area and gave me the opportunity to participate in teaching. Martina provided valuable feedback on my research, and I enjoyed our technical discussions, our walks and undertakings. I enjoy teaching a lot, and I want to thank Richard Küng for mentoring me as part of a didactic course I was taking. Richard involved me in building up his course from scratch. I learned a lot about teaching and writing high-quality lecture notes.

I want to thank Roberto Sebastiani and Andrea Passerini for hosting me during my research stay at the University of Trento. Roberto and I had many intense discussions, and I am grateful for this co-operation.

My thanks go to Adnan Darwiche at the University of California who agreed to evaluate my thesis.

I am grateful to have a supporting family by my side. My husband Frank went with me through both easy and difficult times. Our sons Dario and Silvan encouraged me to pursue my dream of doing a PhD knowing this would mean to live apart. During my studies, and even before starting my PhD, Frank, Dario, and Silvan went for hikes and undertook things without me such that I could work towards my goal during one of the most beautiful winters I can remember.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Part I

INTRODUCTION

1 MOTIVATION AND OVERVIEW

Many tasks relevant to practice may be formulated as propositional formulae. These formulae are defined over variables which represent some statement and can be assigned the logical values true and false in order to express that the corresponding statement does or does not hold. The value of a variable is inverted by applying the negation operator ($\neg$) to it. Variables can be combined to form more complex expressions, such as the formulae mentioned above, by means of logical connectives. In this thesis, we mostly consider the logical connectives conjunction ($\wedge$) and disjunction ($\vee$) as well as equivalence ($\leftrightarrow$). A conjunction of variables is true if all its variables are true. In contrast, a disjunction of variables is true if at least one of its variables is true. Two variables are logically equivalent if they have the same logical value. These logical connectives and the negation operator are similarly applicable to formulae. Finally, the truth value of a formula is determined by assigning its variables their truth value and by recursively propagating these truth values through the connectives.

A satisfiability solver can be used to determine whether the variables of a formula can be assigned values such that the formula evaluates to true under the resulting assignment, or, stated otherwise, whether this formula is satisfiable. This problem is known as the *satisfiability problem of propositional logic*, or SAT. An assignment satisfying a formula is also referred to as *model* of that formula. A formula may have multiple models, and for some tasks one might not only be interested in the satisfiability of the formula (SAT) but in the models themselves (All-SAT) or in their number (#SAT). When referring to All-SAT, we sometimes talk of *(propositional) model enumeration*.[1] Similarly, the term *(propositional) model counting*[2] may be used when referring to #SAT.

A classical application for #SAT is probabilistic reasoning [13, 120, 167, 174]. Other examples include software and hardware verification [26, 36, 73, 74, 104, 201], model-based diagnosis of physical systems [112], planning [10, 202], product configuration [110, 203], and cryptography [106]. Model enumeration is a key task in, e. g., lazy Satisfiability Modulo Theories [176], predicate abstraction [118], software product line engineering [80], model checking [21, 131, 184], preimage computation [119, 180], system engineering [187], and automatic test pattern generation (ATPG) [190]. The breadth of these tasks demonstrates the practical relevance of both All-SAT and #SAT and the need for efficient solving methods.

From a computational point of view, All-SAT and #SAT are harder than SAT [90]. But why is this the case? After all, SAT, All-SAT, and #SAT look not that different. The reason is a simple one. To show that a formula is satisfiable, it is sufficient to

1 The focus in this thesis is on model enumeration *without repetition*, i.e., models must be recorded only once.

2 In this thesis, we refer to *exact* model counting, in contrast to *approximate* model counting.

provide *one single* model. In contrast, in All-SAT and #SAT *all* possible assignments need be checked, and their number is exponential in the number of variables. More precisely, each variable can be assigned one out of two values, and there are $2^3 = 8$ possible assignments for 3 variables, $2^{10} = 1'024$ possible assignments for 10 variables, $2^{100} = 1'267'650'600'228'229'401'496'703'205'376 \approx 1.27 \cdot 10^{30}$ possible assignments for 100 variables, $2^{1'000} \approx 1.07 \cdot 10^{301}$ possible assignments for $1'000$ variables—a number 302 digits long—and so on, and industrial problems may contain millions of variables. To get a better grasp of this huge number, we compare it to the number of particles in the observable universe. This number is estimated to be $4 \cdot 10^{80}$[193], which is way smaller than the number of assignments we need to check. These assignments form the *search space*, whose size is given by their number. When devising a model counting or model enumeration algorithm, one should therefore aim at pruning the search space to be processed at once, i. e., reducing the number of assignments to be checked. This way, SAT solvers are capable of solving formulae with millions of variables.

The state of the art in #SAT solving is to partition the input formula into subformulae, or *components*, defined over pairwise disjoint variable sets. The model count of each component is then computed separately and the results combined. In that manner, the search space to be processed during each computation is reduced, and the individual computations become feasible. This idea was first proposed by Bayardo Jr. and Pehoushek [15] and subsequently improved in several ways. These improvements mostly affect data structures and algorithms for, e. g., caching and selecting components or binary constraint propagation [172, 173, 189]. The fact that the individual components can be processed independently motivated the implementation of a parallel [37] and a distributed [38] algorithm based on this approach, which we sometimes refer to as *component-based reasoning*.

When enumerating the models of a propositional formula, we face similar issues, and similarly to #SAT, in many tasks—and in this thesis—each model must be enumerated exactly once. This can be ensured by adopting the Davis-Putnam-Logeman-Loveland (DPLL) algorithm [60]. This means that whenever a conflict occurs or a model is found, all assignments after the most recent decision[3] are undone, and the decision variable is *flipped*, i. e., assigned its opposite value. The decisions are flipped in reverse assignment order, ensuring hat each assignment is encountered exactly once. The downside of DPLL is that the solver might spend a significant amount of time in a region of the search space containing no model. A decision taken much earlier in the search might lead to a partial assignment[4] which can not be extended to a model, but the solver is unable to detect this fact. Instead, by flipping all decisions after the "bad" one by one it checks all total extensions of the corresponding assignment.

Conflict-driven clause learning (CDCL) [127, 128, 146] enables the solver to detect bad assignments early and to resume the search at an earlier stage with a more promising assignment. This may result in a significant work saving, and most state-of-the-art SAT solvers are based on CDCL. They use specialized algorithms, such as *conflict analysis* for determining an explanation for a conflict and *conflict-driven backjumping* for subsequently fixing the search direction.

3 In a *decision*, a variable is chosen and assigned a value according to some heuristics. This contrasts *propagation*, in which a variable need be assigned a specific value in order to avoid a conflict.

4 In a *partial* assignment not all variables occur. Assignments containing all variables are called *total*. Similarly, *partial* and *total* models are defined.

However, avoiding repetitions is an issue in CDCL-based All-SAT solvers with non-chronological backtracking.[5] State-of-the-art All-SAT solvers therefore either rule out the models already found by adding *blocking clauses* to the formula [104, 131] or adopt chronological backtracking as in DPLL [94]. The first method suffers from a potential exponential growth of the formula slowing down the solver, while the latter prevents escaping search space regions without models early.

In some tasks, such as bounded model checking [184] and predicate abstraction [118], not all variables might be interesting. In this case, the models projected onto these interesting, or *relevant*, variables are sought. Consequently, models differing only in *irrelevant* variables, i. e., variables which are not interesting, are considered the same. This can be achieved by either the use of blocking clauses or by assigning relevant variables before irrelevant ones [94].

These arguments motivated the foci of this thesis, namely pruning the search space, avoiding repetitions, and projecting models in the context of (proposition) model counting and enumeration. They are described in the next paragraph.

AIM OF THIS THESIS. We explore alternative approaches to (projected) propositional model counting and enumeration. One focus is on pruning the search space by detecting partial models. In fact, a partial model can be extended to a total model by arbitrarily assigning the variables which do not occur in it, i. e., are unassigned, and there exist 2^n different total extensions of a partial model with n unassigned variables. These extensions need not be checked explicitly, and the labor saving scales exponentially with n. Partial models therefore provide an efficient means not only to represent sets of total models, namely the ones the partial model can be extended to, but also to reduce the number of assignments to be checked explicitly. One goal is therefore to find short partial models.

Our second focus is on methods ensuring that each model is counted or enumerated exactly once. As pointed out above, repetitions are an issue in #SAT solvers based on CDCL with non-chronological backtracking. However, these solvers are able to escape search regions containing no model early saving a potentially significant amount of work. Our second goal is therefore to avoid repetitions while exploiting the power of CDCL with non-chronological backtracking.

A third focus is on projected model enumeration and enumeration. The goal here is to avoid finding models which differ only in irrelevant variables.

Each focus is addressed in several publications, and for some of them we provide multiple solutions. They are described in the rest of this chapter.

DETECTING PARTIAL MODELS. We present three different approaches for partial model detection. In our first approach, we exploit the fact that modern SAT solvers are able to determine that the input formula evaluates to false under a partial assignment. We also say that this assignment *falsifies* the input formula or that a *conflict occurs* in the input formula. An assignment falsifying a formula is called a *counter-model* of this formula. Notice that it is a model of its negation. Similarly, a counter-model of the negation of a formula is a model of this formula. In other words, if we are given a model of a formula and evaluate the negation of this formula under this assignment, we obtain a conflict. Accordingly, if a conflict in the negation of a formula occurs, the corresponding assignment is a model of

5 Throughout this thesis, the terms *non-chronological backtracking* and *conflict-driven backjumping* are used synonymously.

this formula. This allows us to detect partial models and lays the ground for the first approach presented in this thesis.

The main idea is as follows. We pass both the input formula and its negation to a #SAT solver. The solver maintains one single *trail* which represents a (partial) variable assignment and extends it in an iterative manner. Whenever a conflict in the negation of the formula occurs, the corresponding trail represents a (partial) model of the input formula. We call this approach *dual* and sometimes accordingly talk about *dual reasoning*.

Even shorter partial models might be obtained by determining assignments which *logically entail*[6] the input formula. These assignments are (partial) models of the formula. It turns out that the test capturing the precise entailment condition is computationally expensive. However, sometimes a cheaper—but less powerful—test might be sufficient. We therefore present four entailment checks of different strengths and computational costs, some of which involve dual reasoning.

Finding partial models in a non-dual setting involves some kind of check, either for logical entailment or for satisfiability. These checks are more expensive than just assigning the remaining variables arbitrarily. And this is also the reason why state-of-the-art SAT solvers only detect total models. However, partial models might also be obtained from total models by removing the literals which are not needed in order to satisfy the input formula. This is the task of *model shrinking*, which we address in our most recent work.

For model shrinking we fall back on dual reasoning. Evaluating the total model under the negation of the formula results in a conflict, and subsequent conflict analysis allows for identifying the assignments involved in the conflict. These are exactly the assignments which are sufficient to satisfy the input formula, and we can escape the region of the search space containing only models by simply backtracking to the position in the trail before the most recent decisions which do not contribute to the model.

AVOIDING FINDING MODELS MULTIPLE TIMES. As already mentioned, #SAT solvers based on CDCL with non-chronological backtracking may find models multiple times. Blocking clauses offer a solution to this problem, but they come at a price. To tackle the performance issues due to the use of blocking clauses, we explore two different approaches. Our first idea is to associate with each flipped decision literal the number of models detected at this stage of the search. We refer to this step as *flipping*. Whenever backtracking past a flipped decision occurs, the associated model count is subtracted from the number of models found so far, because these models will be rediscovered later. We call this subtraction *discounting*. While models may still be detected multiple times, flipping and discounting ensure that the correct model count is returned.

A second method to avoid the use of blocking clauses is to combine chronological backtracking with CDCL. This method, in the following referred to as *chronological CDCL*, was introduced by Nadel and Ryvchin [149] in the context of SAT solving. It allows for escaping regions of the search space without models early while remaining in the proximity of the region of the search space currently explored if neither a model has been found nor a conflict has occurred. This makes sense in model enumeration and counting, since the search space need be processed ex-

6 Following Sebastiani [177], a partial assignment logically entails a formula, if all its total extensions satisfy the formula.

haustively. Like CDCL with non-chronological backtracking, chronological CDCL always terminates.

PROJECTING MODELS. The challenge consists in avoiding the detection of models which only differ in irrelevant variables, since they are considered the same. Irrelevant variables can be variables representing some property which is not interesting for the task at hand, or auxiliary variables introduced by the transformation of an arbitrary propositional formula into CNF.

In general, models contain both relevant and irrelevant variables. These models are then projected onto the relevant variables by simply removing the irrelevant ones. If now two models are the same except, e. g., for one irrelevant variable occurring positively in one model and negatively in the other, after projection these models are equal. This situation can occur if an irrelevant variable was decided before all relevant variables were assigned, because this decision is flipped later. Prioritizing decisions on relevant variables over decisions on irrelevant variables ensures that only models differing in at least one relevant variable are found.

Our aim is to detect short projected models. To this end we combine projection with each of dual reasoning, logical entailment, and model shrinking.

2 OUTLINE OF THE THESIS

This thesis encompasses one workshop and four conference papers as well as one article currently under review. Their relationship is visualized in Figure 2.1. The papers are grouped into three parts visualized by the shadowed boxes.

In Part II, we provide background and related work. In Part III, Part IV, and Part V described below, the published versions of the papers are included and discussed separately. For the sake of readability, we unified the notation.

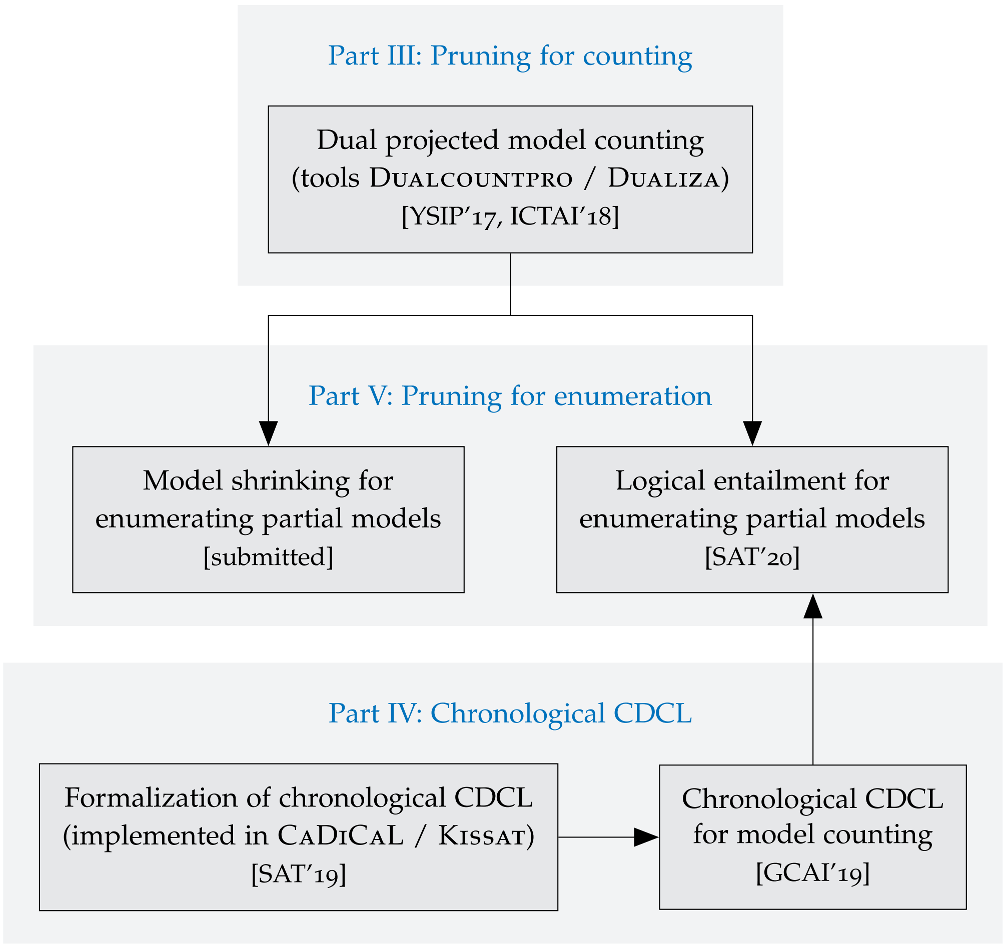

Part III: Pruning for counting
Dual projected model counting (tools DUALCOUNTPRO / DUALIZA) [YSIP'17, ICTAI'18]
Part V: Pruning for enumeration
Model shrinking for enumerating partial models [submitted]
Logical entailment for enumerating partial models [SAT'20]
Part IV: Chronological CDCL
Formalization of chronological CDCL (implemented in CADICAL / KISSAT) [SAT'19]
Chronological CDCL for model counting [GCAI'19]

Figure 2.1: Outline of the thesis.

PART III: DUAL PROJECTED MODEL COUNTING. The main contributions are two formal frameworks for dual model counting. Our first calculus is based on the DPLL algorithm. We also present a formalization of a non-dual variant and argue that our framework is sound. The correctness of our rules is experimentally checked by means of a proof of concept implemented in SWI-Prolog, called DualCountPro. In our second calculus we extend this approach to support projection and by methods widely used in state-of-the-art SAT solvers, such as conflict analysis and conflict-driven backjumping. We extend DualCountPro accordingly and provide a robust and carefully tested implementation in C, called Dualiza, and discuss implementation details. Finally, preliminary experiments with both tools show that dual model counting results in shorter execution traces compared to the ones of its non-dual variant and leads to the detection of short partial models.

This part contains one workshop paper (Chapter 9) and one conference paper (Chapter 11), which are both peer-reviewed. They are discussed separately in Chapter 10 and Chapter 12, respectively. We have implemented a proof of concept in SWI-Prolog. Our tool proved valuable in checking our rules and is presented in Chapter 13. The material in this chapter was not published to date.

PART IV: CHRONOLOGICAL CDCL FOR MODEL COUNTING. We formalize and generalize chronological CDCL [149] and provide a formal proof of correctness. Our calculus is turned into an algorithm establishing the foundation for our implementation of chronological CDCL. Implementation details are discussed, and experimental results are reported. This framework is then extended to support exact unweighted model counting. It is devised as an enumeration approach facilitating a formal proof of its correctness.

The papers contained in this part (Chapter 14 and Chapter 16) were published at conferences and are both peer-reviewed. These papers are discussed separately in Chapter 15 and Chapter 17.

PART V: PARTIAL MODEL ENUMERATION. We combine the techniques introduced in Part III and Part IV for partial model enumeration. The focus in our first calculus is on detecting (partial) assignments which logically entail the input formula. We discuss four entailment checks of different strengths and computational costs. In our second approach, partial models are obtained by shrinking total ones. We present a calculus for projected model enumeration without repetition and a relaxed variant admitting repetitions. For both calculi, algorithms and formal proofs of their correctness are provided.

This part contains one peer-reviewed conference paper (Chapter 18) and an article currently under review (Chapter 20). The latter is not an extended version of a conference paper and hence contains only material which to date is unpublished. In contrast to Chapter 9, Chapter 11, Chapter 14, Chapter 16, and Chapter 18, we have already updated several minor issues on-top of the submitted version as archived in [140]. The conference paper and the submitted article are discussed in Chapter 19 and Chapter 21, respectively.

PART VI: CONCLUSION. The thesis is discussed as a whole and the results of our papers compared. We provide more background concerning dual reasoning. After summarizing our contributions to propositional satisfiability, propositional counting and propositional model enumeration, we conclude by pointing out future research directions.

3 CONTRIBUTIONS

In this chapter, a short high-level summary of each paper contained in this thesis is given, and my own contributions to each of them are pointed out.

3.1 PAPER 1: AN ABSTRACT DUAL PROPOSITIONAL MODEL COUNTER

Our first work [24] published at the *Second Young Scientist's International Workshop on Trends in Information Processing (YSIP2) 2017*, addresses DPLL-based model counting. We assume the input formula F to be an arbitrary propositional formula over a set of variables V. However, most modern SAT solvers work on formulae in conjunctive normal form (CNF), and they implement efficient algorithms tailored to CNF formulae. To take advantage of those algorithms, we transform F into a logically equivalent CNF formula P by, e. g., eliminating double negations and applying the De Morgan's and distributive laws. Since F and P are logically equivalent, they have the same models and model count. In our counting algorithm, we follow the main idea introduced by Birnbaum and Lozinskii [27]: the model count of F equals the sum of the model counts of F after setting a variable $v \in V$ to 1 and the model counts of F after setting v to 0.

We present a calculus based on this idea. It is devised as a state transition system and consists of five rules, one capturing decisions and two for each termination and chronological backtracking, one applicable to satisfying and one applicable to falsifying assignments, respectively. In this paper, both total and partial assignments are called *interpretations*. In *chronological* backtracking, referred to as *naive* backtracking (because it does not take into account the reason of the conflict), the most recent decision literal is flipped.

Motivated by previous work [72, 93], we then extend our counting algorithm to work on CNF representations of both the input formula F and its negation $\neg F$, which we denote by P and N, similar to *positive* formula and *negative* formula, respectively. We assume both P and N to be defined over the *same* set of variables V over which F is defined. This particular condition ensures that whenever an assignment satisfies P, it falsifies N and vice versa. As explained above, having both N and P facilitates the detection of partial models of F, since these are exactly the assignments under which a conflict in N occurs. Since variables can be either true or false, each partial model with n unassigned variables represents 2^n total models. Our DPLL-based model counter takes both P and N as arguments, processes them simultaneously, and computes the model count of P (and hence of F) according to this idea. It works on two formulae which are the negation of each other, hence we call it *dual*.

We restrict our framework to a minimal set of rules, namely decisions, backtracking and termination. This simplifies the presentation since fewer cases have to be distinguished and reasoned about. To make sure that the correct model count is returned by our framework, every possible assignment must be tested exactly once. This is ensured by the simple backtracking mechanism. If our framework is sound, every implementation which can be modeled by means of it is sound as well. This comprises optimizations, such as unit propagation, which is not included in our formalization but can be simulated by a decision and subsequent backtracking.

Finally, we extend both the dual and non-dual calculus by rules capturing unit propagation in P and (in the dual case) N and demonstrate their function by means of an example. The trace produced by the dual calculus turns out to be significantly shorter than the trace of its non-dual version. However, for the latter a shorter trace is obtained by adopting a different decision heuristic. We implemented our framework in SWI-Prolog [199] making use of the PIE system [198].

The main gain in this work is to save decisions by applying unit propagation in N. However, it helped to gain a deeper understanding of the dual approach. To our best knowledge, it is the first dual model counting method. Preliminary experiments confirmed its suitability for propositional model counting.

MY CONTRIBUTIONS. The main idea is motivated by earlier results [27, 72, 93] and adapted for propositional model counting in cooperation with Armin Biere. In the initial brain-storming and discussing phase we were supported by Andreas Fröhlich. I developed the rules underlying both counting algorithms. The idea to simulate unit propagation by means of decisions and backtracking goes back to a discussion with Christoph Wernhard at TU Dresden. I devised first versions of both the dual and non-dual formal framework and refined them in discussions with my co-authors Armin Biere and Steffen Hölldobler as well as my colleagues at TU Dresden, Christoph Wernhard and Tobias Philipp. I drafted the soundness argument and refined it with Armin Biere. I created the examples and their variants and elaborated the rules for unit propagation. The SWI-Prolog implementation is my sole work as well as the preliminary experiments showing the suitability of the approach.[1] A first draft of the paper was done by me and refined in cooperation with Armin Biere and Steffen Hölldobler.

3.2 PAPER 2: DUALIZING PROJECTED MODEL COUNTING

Our second work [137] published at the *IEEE 30th International Conference on Tools with Artificial Intelligence (ICTAI) 2018*, extends the dual approach presented in our former work by projection capabilities and methods widely used in (non-dual) state-of-the-art SAT solvers, such as conflict analysis and conflict-driven backjumping. The input formula F is assumed to be an arbitrary propositional formula defined over the set of variables V. We partition V into the set of *relevant* variables X and the (possibly empty) set of *irrelevant* variables Y. Similarly, we can decompose an assignment into a *relevant* and an *irrelevant* part. The models of F projected onto X then are exactly the relevant parts of the assignments satisfying F.

Without the introduction of fresh variables, but by eliminating double negations and adopting the De Morgan's and distributive laws, the CNF transformation of a formula is exponential in size compared to the original formula. Therefore, as in the Tseitin transformation [192] and the Plaisted-Greenbaum transforma-

1 Due to space requirements, these experiments are not included in the paper.

tion [164], new sets of variables S and T are introduced during the transformation of F and $\neg F$ into CNF formulae P and N, respectively. We call the variables in S *primal* variables and the variables in T *dual* variables. These fresh variables are irrelevant, and the models of F need be projected onto X.

State-of-the-art SAT solvers use conflict-driven clause learning (CDCL) [127, 128, 146], which extends DPLL with a procedure to analyze conflicts and learn a clause representing the reason for the conflict. This clause on the one hand prevents the solver from repeating a bad assignment and on the other hand enables it to continue the search with a more promising assignment.

Our model counter executes a dual variant of CDCL on P and N. The calculus describing it consists of rules capturing termination, backtracking, unit propagation in both P and N, decisions, and the discarding of redundant clauses. Conflicts are handled by either backtracking chronologically or by executing conflict analysis and subsequent conflict-driven backjumping. Our rules handling the detection of a (partial) model are presented in two versions. In one, we remember the models found at the current decision level. This allows us to jump back over branches in which models were found without the need for blocking clauses by just subtracting the corresponding model count from the number of models found so far. In the other, we add a blocking clause to P.

We discuss the combination of the different concepts for handling conflicts and satisfying assignments in the context of dual model counting. To ensure the correctness of our counting method, we first decide the (relevant) variables in X, followed by the (irrelevant) variables in Y and the (primal) variables in S. We never decide the (dual) variables in T. This decision order guarantees that whenever an assignment falsifies N, it can be extended to a total model of P and hence of F.

The SWI-Prolog implementation of the previous work was extended accordingly and proved a valuable tool for checking correctness of the rules. Our new tool DUALIZA implements the calculus. It takes as input an arbitrary propositional formula, a circuit, or a formula in CNF. DUALIZA counts or enumerates models with or without projection and can also act as a simple SAT solver. We present some preliminary experimental results and show the orthogonal strength of DUALIZA compared to component-based #SAT solvers.

MY CONTRIBUTIONS. The definition of the new concepts and the framework is the result of a joint effort of the authors. I extended my SWI-Prolog implementation of the previous work to accommodate the new rule set. This implementation was very helpful for checking (and fixing) the rules. My focus was on the theoretical part of this work, and on the writing of the paper.

3.3 PAPER 3: BACKING BACKTRACKING

In our paper [138] published at *The 22nd International Conference on Theory and Applications of Satisfiability Testing (SAT) 2019*, we formalize and generalize the combination of chronological backtracking with CDCL, also referred to as *chronological CDCL*, introduced by Nadel and Ryvchin [149]. It is motivated by the observation that after backjumping the assignments just undone might be repeated [188]. In chronological CDCL, backtracking after learning a conflict clause therefore does not occur to the assertion level but to the decision level preceding the conflict level, i. e., the decision level at which the conflict occurred, but the literal propagated next is assigned to the assertion level. The idea is to remember the decision level to which backjumping would occur in standard CDCL with conflict-driven backjumping without undoing the assignments at higher decision levels. As a consequence, the literals on the trail are not ordered in ascending order with respect to their decision level anymore, and invariants which to date were considered crucial to CDCL are invalidated. We identify those invariants and show which invariants inherent to CDCL still hold in chronological CDCL.

Our calculus is conceived as a state transition system for SAT. Its rules capture termination, unit propagation, conflict-driven backjumping, and decisions. Unlike in CDCL with conflict-driven backjumping, in chronological CDCL the conflicting clause might contain one single literal at conflict level. This literal is a missed propagation literal due to backtracking chronologically after a conflict. An example for this case is presented and a formal proof of correctness of our calculus provided.

We turn our transition system into an algorithm and provide pseudocode on a higher abstraction level covering exclusively chronological backtracking. Chronological backtracking is added to the CDCL-based SAT solver CaDiCaL. We describe the changes needed to enable chronological backtracking including details on internal data structures not mentioned by Nadel and Ryvchin [149]. These changes are similar to the ones described in their paper [149] and implemented in their solver submitted to the SAT 2018 competition [150].

We evaluate our solver CaDiCaL [19] on the benchmarks from the main track of the SAT Competition 2018. Our experiments confirm the effectiveness of chronological backtracking. Performing exclusively chronological backtracking does not degrade solver performance much and thus, for instance, has potential to be used in propositional model counting.

MY CONTRIBUTIONS. I proposed to investigate chronological CDCL as soon as the corresponding paper [149] was accepted at the SAT 2018 conference. My intuition was that model counting could benefit from the combination of CDCL and chronological backtracking, since it allows to escape regions of the search space without models while remaining in the proximity of the region of the search space currently explored if neither a model has been found nor a conflict has occurred. The presentation of the concepts resulted from discussions with Armin Biere. I developed a first draft of the calculus, which was subsequently refined in many iterations together with Armin Biere. Particularly the rule describing conflict-driven backjumping turned out to be tricky, and we put considerable effort into finding a concise formulation. I created the example explaining the effects of chronological CDCL on the trail and the example for the case where the conflicting clause contains one single literal at conflict level. The proof of correctness is the result of joint work with Armin Biere. The writing was mostly done by me, except for the sections on implementation and experiments.

3.4 PAPER 4: COMBINING CONFLICT-DRIVEN CLAUSE LEARNING AND CHRONOLOGICAL BACKTRACKING FOR PROPOSITIONAL MODEL COUNTING

In our previous work, we identified the potential of CDCL with chronological backtracking for model counting. Developing a calculus for #SAT based on chronological CDCL and providing a formal correctness proof was the goal of our paper [139] published at the *5th Global Conference on Artificial Intelligence (GCAI) 2019*. This work was also presented at the accompanying poster session.[2]

2 https://www.sibyllemoehle.net/images/pdf/MoehleBiere-GCAI19-poster.png

For our counting procedure, we take an enumeration approach, since it facilitates the correctness proof as explained next. Suppose our task is to determine the model count of a propositional formula F. During the execution of our procedure, it must hold anytime that the sum of the number of models already found and the number of models not yet detected equals the model count of F. Now only the first of these three numbers is known, which renders this invariant useless.

Notice that from the trail I representing the current partial assignment we can read off the search space still to be processed. It can be represented as a disjunction of the assignments consistent with I not yet tested. We call it the *pending search space of* I and denote it by $O(I)$. Similarly, the *pending models of* I, i. e., the models consistent with I still to be found, are the models of the formula $O(I) \wedge F$. Our procedure returns a disjunction M, whose disjuncts are the found models.

We extend the calculus developed in our previous work [138] to capture the situation where a model has been found and such that it constructs M as described above. This extension is not dual. It also uses exclusively chronological backtracking, and therefore no measures preventing repetitions need be taken. Chronological backtracking also ensures that the models it finds are pairwise contradicting.

If our calculus is correct, after its termination M is logically equivalent to F, and their model counts coincide. Moreover, since its disjuncts are pairwise contradicting, the model count of M is computed by summing up the number of total assignments represented by its disjuncts. We formally prove that M is logically equivalent to F. Newly introduced invariants allow us first to precisely characterize both the found and pending models and second to show that the union of the two yields the models of F. Instead of using the invariant describe above, we show that it holds anytime that $M \vee O(I) \wedge F$ is logically equivalent to F.

MY CONTRIBUTIONS. I gave the incentive to extend the formalization of chronological CDCL for model counting. I proposed to take an enumeration approach and defined the concepts of pending search space and pending models. The original impulse for these concepts resulted from a discussion with Christoph Wernhard at TU Dresden a few years earlier. He mentioned that from the current trail one could describe the search space not yet tested. I came up with a notation for the pending search space but had no use case for this concept (yet). The concepts I developed were technically correct; their clarity was improved thanks to feedback of Armin Biere. I generated the example demonstrating the notions of pending search space and pending models. I further showed that after flipping the last decision after a conflict, the pending models remain unaltered. I developed the calculus and the proofs. Two new invariants improving the proof structure were introduced in cooperation with Armin Biere. The writing was mostly done by me. The poster accompanying this paper was my sole work. Armin Biere provided feedback on a first draft of the poster. For this poster, I won the *Bolzano Rules and Artificial INtelligence Summit (BRAIN) 2019* Best Poster and Interaction Award.[3]

3.5 PAPER 5: FOUR FLAVORS OF ENTAILMENT

The work [141] published at *The 23rd International Conference on Theory and Applications of Satisfiability Testing (SAT) 2020*, combines the techniques described in all our previous papers. Our main motivation was to find partial models to be used as addends in weighted model integration (WMI) [142, 143]. As their might be

3 https://www.sibyllemoehle.net/images/gcai19certificate.jpg

exponentially many models, the goal was to find *short* partial models in order to reduce their number. The conference took place virtually, and a video had to be prepared in place of a live presentation.[4].

In our work, we laid the focus on detecting partial assignments which *logically entail* the input formula. While every total extension of such an assignment is a model of the input formula, this need not be the case for this assignment itself. Consider as an example the formula $F = (a \wedge b) \vee (a \wedge \neg b)$ over the set of variables $V = \{a, b\}$ and the partial assignment $I = a$, i. e., the assignment in which a is set to true. By evaluating F under I, we obtain the formula $(b \vee \neg b)$ which is undefined. However, both total extensions of I, namely ab and $a\neg b$, are models of F. Obviously, $(b \vee \neg b)$ is valid. But it syntactically differs from true, and the *syntactic satisfiability check* performed by modern (All-)SAT solvers can not detect that F is valid under the partial assignment a.

To ensure the correctness of the integration, repetitions must be avoided. Our main enumeration engine is therefore based on chronological CDCL, and no blocking clauses are needed. Similarly to our previous work, it returns a formula which is logically equivalent to the input formula. In addition, projection is supported.

The basic idea of our procedure is as follows. Assume that at a certain point in the search all enforced variables are assigned, no conflict has occurred, and there are still unassigned variables. A state-of-the-art All-SAT solver now would take a decision. Instead of taking the decision, we propose to first test whether the current assignment already entails the input formula. If this is the case, the enumerator records this assignment and flips the last decision.

The check of the entailment condition under projection is powerful yet computationally expensive. However, sometimes a less expensive—and less powerful—test might do the job. We therefore present the entailment test in four flavors of different strengths and computational costs, two of which make use of dual reasoning. Finally, we discuss examples requiring entailment tests of different strengths.

MY CONTRIBUTIONS. The basic idea was developed in cooperation with Roberto Sebastiani and worked out by me. Together we refined my drafts of the algorithm and the formalization. The entailment checks are the outcome of discussions with Roberto Sebastiani and Armin Biere. This work resulted from my research stay at the University of Trento, from 9th September 2019 to 13th December 2019, enabled by an *AIxIA Income Mobility Grant 2018* offered by the Italian Association for Artificial Intelligence (AIxIA). The research proposal with which I won one out of two grants for incoming researchers was written by me. Roberto Sebastiani provided feedback on my first version. A first draft of the paper was done by me and refined in cooperation with Roberto Sebastiani and Armin Biere. The video accompanying the paper is my sole work. Armin Biere and Roberto Sebastiani provided feedback on its first version.

3.6 PAPER 6: ON ENUMERATING SHORT PROJECTED MODELS

In our work currently under review and available on arXiv [140], partial models are obtained by shrinking total ones using dual reasoning. As in previous papers, P and N denote CNF representations of the input formula F and its negation $\neg F$, respectively. The idea is to call a second SAT solver on $m \wedge N$, where m denotes the (total) model of P we want to shrink. Since m is a counter-model of N,

4 https://www.youtube.com/watch?v=_QBJ3plWt9Y&t=1s

a conflict is obtained. Conflict analysis returns the part of m participating in the conflict, which is exactly the part of m needed to satisfy P and therefore constitutes the shortened model. We are interested in its projection onto the relevant variables, e. g., for removing from m the auxiliary variables introduced during CNF transformations. The projected model is then used to determine the backtrack level.

Our enumeration algorithm is based on CDCL with conflict-driven backjumping and is presented in two variants. In the first variant, enumerating models multiple times is avoided by the use of blocking clauses. By the addition of blocking clauses, P is altered, and dual model shrinking requires these changes be reflected in N. In order to meet this requirement, we propose a dual encoding blocking clause encoding. Importantly, blocking clauses provide the reason for the decision flipped after finding a model fulfilling the assumption of CDCL that the reason of every literal which is not a decision literal is contained in the formula. The addition of blocking clauses therefore ensures a correct functioning of CDCL in the context of model enumeration without repetitions (and therefore model counting).

In the second variant of our enumeration procedure, we relax the uniqueness constraint and admit enumerating models multiple times. Due to the absence of blocking clauses, CDCL fails if a decision flipped after finding a model m is involved. We fix this issue by annotating this flipped decision ℓ with $\neg m$. The clause $\neg m$ provides the reasons for ℓ, but it is not added to P. Furthermore, any clause learned during conflict analysis involving $\neg m$ is logically entailed by $P \wedge \neg m$ but not necessarily by P, to which it is added. For this reason, $\neg m$ can not be used for unit propagation.

For both enumeration methods, algorithms, calculi, and formal proofs of their correctness are provided. We also show their working by examples and sketch the modifications necessary to obtain a generalization to partial model detection.

We present the definitions of soundness and completeness in the context of both partial and total model enumeration. These definitions are inspired by the ones formulated by Christoph Weidenbach [197] in the context of SAT solving.

my contributions. The work presented in this paper was also initiated during my research stay in Trento in 2019. The basic idea and the dual blocking clause encoding was developed in cooperation with Roberto Sebastiani. The generalization to model enumerating with repetitions was proposed by Armin Biere and elaborated by me. The formalizations were done by me, and the idea to annotate flipped literals in the absence of blocking clauses goes back to a discussion with Mathias Fleury. The examples and proofs are my sole work. For these proofs, I developed a mapping from states to lists inspired by Mathias Fleury's master thesis [78]. This mapping turned out to be very useful in proving termination. Mathias Fleury also pointed me to Christoph Weidenbach's definitions of soundness and completeness. I adapted these definitions to model enumeration and refined and precised them in discussions with Armin Biere and Roberto Sebastiani. I did a first draft of the paper. Roberto Sebastiani provided feedback.

4

TOPICS BEYOND THE SCOPE OF THIS THESIS

The focus in this thesis is on SAT-based exact unweighted model counting and irredundant (partial) model enumeration, i. e., (partial) model enumeration without repetitions. There are other variants of model counting, which we did not touch upon. These include weighted model counting and methods exploiting some structural property of the input formula, approximate model counting, and knowledge compilation. We also did only consider existing work in preprocessing and inprocessing. In this chapter, we give high-level presentations of these research areas and show how they are related to this thesis.

4.1 WEIGHTED MODEL COUNTING

A Bayesian network can be expressed as a CNF formula K representing a propositional knowledge base over weighted variables. Reasoning in this network involves summing up the weights of all models of K, possibly given some evidence. The weight of a model is defined as the product of the weights of its literals, and the weight of a literal is defined based on the weight of its variable. Weighted model counting is relevant in artificial intelligence and therefore an active research field [11, 13, 43, 44, 48, 63, 64, 76, 107, 174].

Unweighted model counting can be considered a special case of weighted model counting, namely the one in which all literals are assigned the same weight. By adding weights to the variables of the input formula, our counting (and enumeration) approaches can readily be adapted to support weighted model counting.

4.2 STRUCTURE-AWARE MODEL COUNTING

Although SAT is **NP**-complete, modern SAT solvers are able to solve hard instances stemming from real-world applications. It is assumed that they exploit a hidden structure in these instances [169]. An interesting question is how this hidden structure impacts the complexity of model counting algorithms. The structure of a formula can, for instance, be described by various properties which can be derived from a graph associated with the input formula, and the structural features of industrial benchmarks are studied in several publications [5–7, 130]. The complexity of #SAT is then discussed for classes of formulae whose associated graph is subject to some structural restrictions.

Two structural properties of interest are tree-width and clique-width. They are related, in that a graph having bounded tree-width also has bounded clique-width, while the converse does not hold in general [50]. It has been shown that if the tree-width or the (symmetric) clique-width are bounded, then #SAT is solvable in polynomial time [75–77, 82, 170, 181, 182].

One property which seems to be important in practice is decomposability (see Chapter 1 and Chapter 6). A formula is decomposable, or can be partitioned into components, if it can be partitioned into subformulae over pairwise disjoint sets of variables. Several #SAT solvers are based on this paradigm [15, 172, 179, 189]. Our approaches do not depend on it, since the aim of this thesis was to explore alternative model counting strategies.

Other structural properties studied in the context of model counting are hypergraph, community structure, and centrality [30, 41, 42, 81] as well as symmetry breaking [194]. The interested reader is referred to Chapter 17, *Fixed-Parameter Tractability*, in the *Handbook of Satisfiability* [171].

4.3 APPROXIMATE MODEL COUNTING

For some tasks, an approximate model count is sufficient. Examples are probabilistic reasoning [167], probabilistic planning [62], and machine learning [152]. One approach to approximate model counting consists in sampling (uniformly or non-uniformly) from the set of models of the considered formula, in order to obtain an approximation of its model count [86, 102]. A variety of approximate model counters implement this paradigm, some providing upper and/or lower bounds with or without guarantees [45, 46, 85, 86, 102, 105, 111, 196].

The set of models can also be roughly cut in half by adding a random XOR constraint to the formula. By the addition of n XOR constraints the solution space is partitioned into 2^n subsets, and so on. The partitioning is truly random if each variable occurs with a probability of 50% in these XOR constraints. If after the addition of n XOR constraints the resulting formula becomes unsatisfiable, we can conclude that its model count is at least of the order of 2^n. Notice that the error introduced in the approximation depends on the size of the added XOR constraints, i. e., the shorter the XOR constraints, the greater the error. However, long XOR constraints are hard to deal with for a #SAT solver, hence there is a trade-off between precision and feasibility. Several approximate model counters providing guarantees are based on this idea, some of which in addition provide lower and upper bounds [1, 68, 87, 88, 183, 205].

There exist other approximate model counters which implement neither of the two paradigms mentioned above [69, 109, 194]. Like weighted model counting, approximate model counting is a very active research field. For further details, we refer the interested reader to Chapter 26, *Approximate Model Counting*, in the second edition of the Handbook of Satisfiability [47].

4.4 KNOWLEDGE COMPILATION

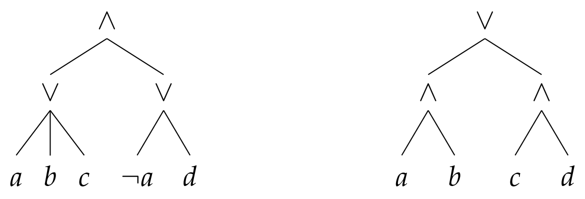
Tackling computationally expensive problems was the motivation for developing the knowledge compilation (KC) paradigm [39]. Examples are equivalence and clausal entailment checks or model counting and model enumeration. The idea is to translate a formula from one language into another in which the task of interest can be executed in polynomial time [59]. The knowledge compilation map [59] contains an in-depth discussion of such languages and their properties, and other (families of) languages have been introduced since its publication [56, 70, 108].

In this thesis, we did not consider KC in the first place. However, in our model counting and enumeration methods based on chronological CDCL [139–141], we generate a *disjoint Sum-of-Products (DSOP)* formula, i. e., a DNF whose disjuncts are pairwise contradicting, which is logically equivalent to the input formula. The models of a DSOP formula are its disjuncts, and its model count equals the sum of the number of total assignments represented by them. Therefore, formulae in DSOP support both model enumeration and model counting in polynomial time. In this sense, our approaches can be considered related to KC. Unlike in the work by Huang and Darwiche [99, 100], where the translated formula is obtained by recording the trace of a DPLL execution as a graph, our DSOP formula is generated by an exhaustive search and combining the (partial) models of the input formula disjunctively. Due to its relatedness to our work, we give a slightly more detailed presentation of KC based on the article by Darwiche and Marquis [59].

A propositional formula F over a set of variables V is said to be in *Negation Normal Form (NNF)*, if it is built from literals with variable in V and the logical connectives disjunction ($\vee$) and conjunction ($\wedge$) [123]. If in addition the conjuncts of each conjunction are defined over disjoint sets of variables, the formula F is said to be in *Decomposable Negation Normal Form (DNNF)* [51, 52]. This *decomposability property* is essential for the component-based model counting approach presented in Chapter 6 and Example 6.3. The Disjunctive Normal Form (DNF), which is a disjunction of cubes which are conjunctions of literals, meets the decomposability property and is a subset of DNNF [51, 52], while the Conjunctive Normal Form CNF, which is a conjunction of clauses which are disjunctions of literals, is a subset of NNF. Each NNF formula can be represented as a *rooted, directed acyclic graph (DAG)*, in which a leaf node represents either a literal or a truth value and in which all internal nodes and the root node are label with either $\vee$ or $\wedge$. Both CNF and DNF meet the *flatness* property: the corresponding DAG has height at most two. Following the definition in the knowledge compilation map [59], the language CNF satisfies the *simple-disjunction* property saying that the literals in each clause are leafs sharing no variable, whereas the language DNF meets the *simple-conjunction* property, i. e., the literals in each cube are leaves sharing no variable. These properties in particular forbid tautologies, i. e., disjunctions of the form $(v \vee \neg v)$, and contradictions, i. e., conjunctions of the form $(v \wedge \neg v)$, where $v \in V$. The terminology is clarified by a basic example.

Example 4.1 (Negation Normal Forms)**.** *We consider the set of propositional variables $V = \{a, b, c, d\}$ and two formulae F and G defined over the variables in V. The formula $F = C_1 \wedge C_2 = (a \vee b \vee c) \wedge (\neg a \vee d)$ defined over V is in NNF: it is composed of literals and the logical connectives conjunction and disjunction. However, it is not in DNNF, since $\mathsf{var}(C_1) \cap \mathsf{var}(C_2) = \{a\} \neq \emptyset$. It meets the simple-disjunction property, and its DAG representation, depicted on the left hand side in Figure 4.1, has height two. Therefore, the formula F is in CNF. Similarly, the formula $G = D_1 \vee D_2 = (a \wedge b) \vee (c \wedge d)$ is in NNF. Since the conjuncts in both D_1 and D_2 are defined over different sets of variables, i. e., $\mathsf{var}(a) \cap \mathsf{var}(b) = \emptyset$ and $\mathsf{var}(c) \cap \mathsf{var}(d) = \emptyset$, it is also in DNNF. Furthermore, it meets the simple-conjunction property and its DAG representation, visualized on the right hand side in Figure 4.1, has height two. Therefore, the formula G is in DNF.*

A formula in which the disjuncts of each disjunction are pairwise contradicting, is said to be *deterministic*, and if all disjuncts contain the same variables, it is said to be *smooth* [53]. Formulae in NNF satisfying both decomposability and determinism are in *deterministic Decomposable Normal Form (d-DNNF)*. A formula



$F = (a \vee b \vee c) \wedge (\neg a \vee d)$ $\qquad$ $G = (a \wedge b) \vee (c \wedge d)$

Figure 4.1: DAG representation of CNF (left hand side) and DNF (right hand side).

in DNF which is deterministic and smooth is in the language *MODS*. Its disjuncts are exactly its total models, and the number of its disjuncts equals its model count. In contrast, the language DSOP mentioned above is deterministic but need not be smooth, which makes it a suitable target language for the approaches developed in this thesis. In fact, its cubes are possibly partial satisfying assignments and therefore allow for a more compact representation of a formula compared to MODS. Both MODS and DSOP support model counting and irredundant model enumeration in polynomial time. The relation between determinism and smoothness and the model count of a formula is explained by means of an example.

Example 4.2 (Determinism, smoothness, and model count)**.** *Consider the DNF formula $F = (a \wedge b \wedge c) \vee (a \wedge b \wedge \neg c) \vee (\neg a \wedge \neg b \wedge c) \vee (\neg a \wedge b \wedge c)$ defined over the set of propositional variables $V = \{a, b, c\}$. The formula F is deterministic and smooth. In fact, its disjuncts are pairwise contradicting and each disjunct contains all variables, and therefore F is in the language MODS. Its total models are exactly the total assignments represented by each disjunct, hence $\mathsf{models}(F) = \{abc, ab\neg c, \neg a\neg bc, \neg abc\}$, and $\#F = 4$. The DNF formula $G = (a \wedge b) \vee (\neg a \wedge c)$ defined over V is deterministic but not smooth, and G is in DSOP. Each of its disjuncts represents a set of two total models of G, and due to the determinism property, these sets are disjoint. The total models of G are given by $\mathsf{models}(G) = \{abc, ab\neg c, \neg a\neg bc, \neg abc\} = \mathsf{models}(F)$. The formulae F and G are logically equivalent but G containing two cubes and four literals is more compact than F, which consists of four cubes and 12 literals.*

The trace of an exhaustive DPLL search represents a DAG and can be considered the compilation of a formula from CNF into d-DNNF [99, 100]. This is similarly the case for the traces of dynamic programming algorithms for #SAT [31]. There are other languages of interest in the context of model counting or model enumeration, which are not relevant in this thesis. The aim of this paragraph is to provide pointers to further information. The language *Decision-DNNF* is defined similarly to d-DNNF but with so-called *decision nodes* in place of the deterministic disjunction nodes [114, 159]. A decision node is a disjunction node whose disjuncts are of the form $v \wedge \alpha$ and $\neg v \wedge \beta$, where v is a variable and α and β are subgraphs representing subformulae. Other languages are *Sentential Decision Diagrams (SDD)* [56], which are a subset of Decision-DNNFs, and *(Extended) Affine Decision Trees (EADT)* [108]. Finally, there exist structured versions of DNNF and d-DNNF [163]. Due to its relevance in many research areas and tasks, including proof systems and reasoning, as well as applications in artificial intelligence, such as multi-agent path finding and planning, research in KC is active for more than two decades [3, 16, 31, 39–41, 59, 79, 121, 162, 163, 175, 178], and a variety of knowledge compilers are available [53–55, 108, 114, 117, 147, 148, 159–161, 186, 191].

4.5 PREPROCESSING AND INPROCESSING

Both preprocessing and inprocessing are powerful tools and are extensively used in SAT solving [14, 25, 49, 66, 71, 133, 151, 158]. Some of these methods are only satisfiability-preserving, i. e., they do not transform a satisfiable formula into an unsatisfiable one and vice versa. But they are not equivalence-preserving, i. e., they alter the models and usually also the model count of the processed formula. Consequently, in All-SAT and #SAT, only equivalence-preserving pre- and inprocessing techniques must be used. Due to the hardness of All-SAT and #SAT, more expensive techniques might pay off. In our implementations, we fell back on existing work on preprocessing for model counting [113, 115].

Part II

BACKGROUND

5

PROPOSITIONAL SATISFIABILITY

Let V be a set of *propositional variables*. These are variables interpreted over the Boolean constants $\mathbb{B} = \{0, 1\}$. As common in the literature, 0 denotes false, and 1 denotes true. A *literal* ℓ is a variable $v \in V$ or its negation $\neg v$. Its variable is obtained by $\mathsf{var}(\ell) = v$, if $\ell = v$ or $\ell = \neg v$. A *propositional formula* $F(V)$ defined over V is composed of the truth values 0 and 1, literals with variable in V, the negation operator ($\neg$), and the logical connectives conjunction ($\wedge$), disjunction ($\vee$), implication ($\rightarrow$), and equivalence ($\leftrightarrow$). We also might write F as a shortcut for $F(V)$.

A *total assignment* $\sigma \colon V \mapsto \mathbb{B}$ maps the variables in V to the truth values 0 and 1. It can be applied to F to yield its truth value $\sigma(F) \in \mathbb{B}$, also referred to as the *truth value of F under σ*. The *satisfiability problem of propositional logic*, or SAT, is the task of determining whether there exists an assignment σ to the variables in V such that F evaluates to 1 under this assignment. If $\sigma(F) = 1$, we say that σ *satisfies* F, or that F is *satisfiable*, and call σ a *model* of F. If instead $\sigma(F) = 0$, we say that σ *falsifies* F and call σ a *counter-model* of F. If $\sigma(F) = 0$ for all possible assignments σ, we say that F is *unsatisfiable*.

Example 5.1 (Propositional formula and satisfiability)**.** *Let $F = (a \wedge c) \vee (b \wedge d)$ be a propositional formula over the set of variables $V = \{a, b, c, d\}$. The total assignment $\sigma = \{a \mapsto 1, b \mapsto 0, c \mapsto 1, d \mapsto 1\}$ is a model of F. There are others, but for showing that F is satisfiable, providing one model is sufficient.*

Modern SAT solvers mostly work on formulae in *conjunctive normal form (CNF)*. In propositional logic, a CNF formula is a conjunction of *clauses*, which are disjunctions of literals. In principle, a CNF representation of F can be obtained by eliminating double negations and applying the distributive and De Morgan's law. However, this method might lead to a substantial growth of F.

To avoid this blowup, a CNF representation of F can be computed by means of either the Tseitin transformation [192] or the Plaisted-Greenbaum transformation [164]. Both transformations are *satisfiability-preserving*: if F is (un-)satisfiable, its CNF transformation is (un-)satisfiable as well, which is sufficient in the context of SAT solving. Both the Tseitin and the Plaisted-Greenbaum transformation introduce fresh variables, called *Tseitin variables* in the former, for the subformulae of F, and the increase in size of F is at most polynomial in both methods. Next, we explain the working of the Tseitin transformation by an example.

Example 5.2 (Tseitin transformation)**.** *Consider again Example 5.1. The subformulae of $F = (a \wedge c) \vee (b \wedge d)$ are $(a \wedge c)$, $(b \wedge d)$, and $(a \wedge c) \vee (b \wedge d) = F$. Each of these subformulae is represented by a fresh variable. We introduce the Tseitin variables t_1, t_2, and t_3, which are defined as follows: $t_1 \leftrightarrow a \wedge c$, $t_2 \leftrightarrow b \wedge d$, and $t_3 \leftrightarrow t_1 \vee t_2$. These definitions* *are transformed into CNF and combined conjunctively yielding an equisatisfiable CNF representation of F:*

$$\begin{aligned}\mathsf{tseitin}(F) &\overset{\text{def}}{=} (t_1 \leftrightarrow a \wedge c) \wedge (t_2 \leftrightarrow b \wedge d) \wedge (t_3 \leftrightarrow t_1 \vee t_2)\\ &\equiv (\neg t_1 \vee a) \wedge (\neg t_1 \vee c) \wedge (t_1 \vee \neg a \vee \neg c) \wedge\\ &\quad (\neg t_2 \vee b) \wedge (\neg t_2 \vee d) \wedge (t_2 \vee \neg b \vee \neg d) \wedge\\ &\quad (\neg t_3 \vee t_1 \vee t_2) \wedge (t_3 \vee \neg t_1) \wedge (t_3 \vee \neg t_2).\end{aligned}$$

We base the methods we are going to develop in this thesis on the following two SAT solving methods: the *Davis-Putnam-Logemann-Loveland (DPLL) algorithm* [60] and *conflict-driven clause learning (CDCL)* [127, 128]. In the following, we present the core of these methods and provide examples.

The DPLL algorithm is a backtracking-based search algorithm. It is defined recursively, but DPLL-based SAT solvers implement an iterative variant in order to avoid stack overflows. They keep track of the variable assignments and the order in which they occurred by maintaining a *trail*, which we denote by $I = \ell_1 \dots \ell_n$. This trail is a sequence of literals with mutually exclusive variable ($\mathsf{var}(\ell_i) \neq \mathsf{var}(\ell_j)$ for $i \neq j$) and represents a (partial) assignment, i. e., an assignment in which not all variables may occur. Trails and literals can be concatenated, written IJ and $I\ell$, provided $\mathsf{var}(I) \cap \mathsf{var}(J) = \emptyset$ and $\mathsf{var}(I) \cap \mathsf{var}(\ell) = \emptyset$. We denote with $V - I$ the unassigned variables in V, i. e., $V - I = V \setminus \mathsf{var}(I)$.

The *residual of F under the trail I*, denoted by $F|_I$, is obtained by replacing the literals in F occurring on I by 1 and their negation by 0. The residual is defined similarly for clauses and literals. If F is in CNF, its residual under I is obtained by removing from F all clauses containing some $\ell \in I$ and removing from the remaining clauses all occurrences of $\neg\ell$. A clause in which all literals are assigned the value 0 is called *empty clause*. Similarly, a formula in CNF from which all clauses have been removed is called *empty formula*. We denote the empty clause by 0 (false) and the empty CNF formula by 1 (true).

In line with the focus of this thesis, which is formalization, we present the DPLL algorithm as a state transition system with transition relation $\leadsto_{\mathsf{DPLL}}$. Non-terminal states are represented by the pair (F, I), where F denotes a propositional formula in CNF defined over a set of variables V, and I represents a trail over V. The initial state is (F, ε), where ε denotes the empty trail. The terminal state is either SAT or UNSAT, depending on whether F is determined to be satisfiable or unsatisfiable. The transition relation $\leadsto_{\mathsf{DPLL}}$ is the union of transition relations $\leadsto_{\mathsf{R}}$, where R is either DPLLU, DPLLD, DPLLB, DPLLE0, or DPLLE1. The rules describing the core of DPLL are depicted in Figure 5.1. They address unit propagation (rule name suffix U), decisions (D), chronological backtracking (B), and termination either with a counter-model (E0) or with a model (E1). In the following paragraphs, we provide a high-level description of our DPLL framework and introduce related concepts and notation used throughout this thesis.

A DPLL-based SAT solver executes the following steps until either a satisfying assignment is found or F is determined to be unsatisfiable. It computes the residual of F under I. If there exists a clause $C \in F$ in which all literals but one are assigned the value 0, i. e., $C|_I = (\ell)$, the literal ℓ is assigned the value 1, since this is the only way to extend I to a model of F, if possible at all. This is the *unit propagation rule* (DPLLU), and (ℓ) and ℓ are called *unit clause* and *unit literal*, respectively. We refer to ℓ as a *propagation literal* saying that it was *propagated* and call C

DPLLU: $(F, I) \rightsquigarrow_{\mathsf{DPLLU}} (F, I\ell^C)$ if exists $C \in F$ with $(\ell) = C|_I$

DPLLD: $(F, I) \rightsquigarrow_{\mathsf{DPLLD}} (F, I\ell^d)$ if $F|_I \neq 0$ and $\mathsf{units}(F|_I) = \emptyset$ and $\mathsf{var}(\ell) \in V - I$

DPLLB: $(F, I) \rightsquigarrow_{\mathsf{DPLLB}} (F, J\overline{\ell})$ if exists $C \in F$ with $J\ell^d K \overset{\text{def}}{=} I$ and $C|_I = 0$ and $\mathsf{decs}(K) = \emptyset$

DPLLE0: $(F, I) \rightsquigarrow_{\mathsf{DPLLE0}}$ UNSAT if exists $C \in F$ with $C|_I = 0$ and $\mathsf{decs}(I) = \emptyset$

DPLLE1: $(F, I) \rightsquigarrow_{\mathsf{DPLLE1}}$ SAT if $F|_I = 1$

Figure 5.1: DPLL rules.

the *reason* of ℓ. Propagated literals are annotated by their reason,[1] e. g., ℓ^C, and by $\mathsf{units}(F)$ and $\mathsf{units}(F|_I)$, the sets of the unit literals in F and $F|_I$ are obtained.

If $F|_I$ contains no unit clause and not all variables are assigned, an unassigned variable $v \in V$ is chosen and assigned a value according to some heuristic. This is the *decision rule* (rule DPLLD), and the literal ℓ with $\mathsf{var}(\ell) = v$ is called *decision literal* or simply *decision*. We mark the decision literals on I by a superscript, i. e., ℓ^d, and the set consisting of the decision literals on I is given by $\mathsf{decs}(I) = \{\ell \mid \ell^d \in I\}$. The DPLLD rule is applicable only if I does not already falsify F, since this case is dealt with in rules DPLLB and DPLLE0 presented next.

If there exists a clause $C \in F$ such that $C|_I = 0$, also $F|_I = 0$. We say that a *conflict* (in F) has occurred and call C the *conflicting clause*. If I contains some decision literal, not all assignments have been checked yet. The solver *backtracks chronologically*, i. e., it undoes all assignments after the most recent decision ℓ^d and *flips* ℓ by assigning it its *complement* $\overline{\ell} = \neg\ell$ (rule DPLLB). If I contains no decision literal, all assignments have been tested, the formula F is unsatisfiable, and the solver terminates in state UNSAT (rule DPLLE0).

If $F|_I = 1$, the trail I is a model of F. If not all variables occur in I, it is called a *partial* model. Since F is found to be satisfiable, the search terminates in state SAT (rule DPLLE1). Usually SAT solvers refrain from carrying out satisfiability checks but know to have found a model if all variables are assigned and no conflict occurred. This behavior is not reflected in our rules. We explain the working of our DPLL framework on an example pointing out its main drawback.

Example 5.3 (DPLL search)**.** *Let $F = C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5 \wedge C_6 = (a \vee b \vee c) \wedge (a \vee \neg b \vee c) \wedge (\neg a \vee \neg b \vee \neg c) \wedge (\neg a \vee b \vee \neg c) \wedge (b \vee c) \wedge (\neg b \vee c)$ be a formula over the set of variables $V = \{a, b, c\}$. The execution steps are listed in Table 5.1. Suppose a is assigned the value 1 by a decision (step 1), followed by deciding $\neg c$, i. e., assigning c the value 0 (step 2). The residual of F under the trail $I = a^d \neg c^d$ is $F|_I = (b) \wedge (\neg b)$. Propagating b with reason C_5 (step 3) results in clause C_6 becoming empty, and the solver backtracks. The last decision on the trail, $\neg c^d$, is flipped (step 4) followed by propagating b with reason C_4 (step 5). Again, a conflict is reached (C_3 becomes empty), and the*

1 Annotating propagated literals is not needed in DPLL, but it enhances traceability of the example.

Table 5.1: DPLL execution trace for Example 5.3.

STEP	RULE	I	$F\|_I$
0		ε	F
1	DPLLD	a^d	$(\neg b \vee \neg c) \wedge (b \vee \neg c) \wedge (b \vee c) \wedge (\neg b \vee c)$
2	DPLLD	$a^d \neg c^d$	$(b) \wedge (\neg b)$
3	DPLLU	$a^d \neg c^d b^{C_5}$	0
4	DPLLB	$a^d c$	$(\neg b) \wedge (b)$
5	DPLLU	$a^d c b^{C_4}$	0
6	DPLLB	$\neg a$	$(b \vee c) \wedge (\neg b \vee c) \wedge (b \vee c) \wedge (\neg b \vee c)$
7	DPLLD	$\neg a \neg c^d$	$(b) \wedge (\neg b) \wedge (b) \wedge (\neg b)$
8	DPLLU	$\neg a \neg c^d b^{C_5}$	$0 \wedge 0$
9	DPLLB	$\neg a c$	1
10	DPLLE1	$\neg a c$	1

only remaining decision literal, a^d, is flipped (step 6). The resulting residual of F under I contains no unit, and decision $\neg c^d$ is taken (step 7), followed by propagating b with reason C_5 (step 8), after which C_2 and C_6 become empty. After flipping the most recent and only decision on the trail, $\neg c^d$, the model $\neg a c$ is found (step 9). All assignments have been tested, and the search terminates in state SAT *(step 10).*

While in our DPLL framework the most recent decision is always involved in the conflict, it might be the case that many decisions taken previously are not. Concretely, no model can be found no matter how these decision literals are assigned, and—even more interestingly—the same conflict would have been obtained if these decisions would not have been taken at all. The SAT solver could have backtracked earlier saving a potentially large amount of work. A DPLL-based SAT solver can not determine this fact. Instead it flips all decision literals in their reverse assignment order one by one, which in particular includes the decision literals not involved in the conflict. We say that the SAT solver *can not escape regions of the search space without satisfying assignments early*.

This has become evident already in our small Example 5.3. The decision literal a^d did not participate in the conflict occurred after propagating b in step 3: the conflicting clause is C_6, which does not contain $\neg a$. If we would have taken the decision $\neg c^d$ in step 1, we would have obtained the same conflict at step 2 instead of 3, and after backtracking, I would have been c instead of $a^d c$.

Avoiding checking assignments which are known to lead to conflicts is the aim of conflict-driven clause learning (CDCL). It allows to undo not only the most recent but multiple decisions and the resulting propagations in one step, referred to as *non-chronological backtracking* or *(conflict-driven) backjumping* and provides the basis for most modern SAT solvers. In CDCL with non-chronological backtracking, the trail is partitioned into subsequences of literals between decisions in which all literals have the same decision level. Each subsequence starts with a decision literal and extends until the last literal before the next decision. Literals assigned before any decision may form an additional subsequence at decision level zero. The first decision and the resulting propagation literals are assigned decision level one, and so on. Backjumping always occurs right before a decision literal.

$$\begin{array}{ll}
\mathsf{CDCLU:} & (F, I) \rightsquigarrow_{\mathsf{CDCLU}} (F, I\ell^{C}) \ \text{ if } \ \text{exists } C \in F \ \text{ with } \ (\ell) = C|_I \\[2ex]
\mathsf{CDCLD:} & (F, I) \rightsquigarrow_{\mathsf{CDCLD}} (F, I\ell^{d}) \ \text{ if } \ F|_I \neq 0 \ \text{ and } \ \mathsf{units}(F|_I) = \emptyset \ \text{ and} \\
 & \mathsf{var}(\ell) \in V - I \\[2ex]
\mathsf{CDCLJ:} & (F, I) \rightsquigarrow_{\mathsf{CDCLJ}} (F \wedge D, J\ell^{D}) \ \text{ if } \ \text{exists } C \in F \ \text{ with} \\
 & JK \overset{\text{def}}{=} I \ \text{ and } \ C|_I = 0 \ \text{ and } \ (\ell) = D|_J \ \text{ and } \ \ell|_K = 0 \ \text{ and } \ F \models D \\[2ex]
\mathsf{CDCLE0:} & (F, I) \rightsquigarrow_{\mathsf{CDCLE0}} \mathsf{UNSAT} \ \text{ if } \ \text{exists } C \in F \ \text{ with } \ C|_I = 0 \ \text{ and} \\
 & \mathsf{decs}(I) = \emptyset \\[2ex]
\mathsf{CDCLE1:} & (F, I) \rightsquigarrow_{\mathsf{CDCLE1}} \mathsf{SAT} \ \text{ if } \ F|_I = 1
\end{array}$$

Figure 5.2: CDCL rules.

We present CDCL as a state transition system with transition relation $\rightsquigarrow_{\mathsf{CDCL}}$. Like in our DPLL framework, non-terminal states are represented by pairs (F, I) with F and I denoting a propositional formula and a trail over the variables V. The initial state is given by (F, ε), and the terminal states are SAT and UNSAT. The transition relation $\rightsquigarrow_{\mathsf{CDCL}}$ is the union of transition relations $\rightsquigarrow_{\mathsf{R}}$, where R is either CDCLU, CDCLD, CDCLJ, CDCLE0, or CDCLE1. The rules describing CDCL are listed in Figure 5.2. As in our DPLL framework, the name suffixes denote unit propagation (U), decisions (D), termination with counter-model (E0), and termination with model (E1). The suffix J refers to backjumping and the corresponding rule CDCLJ replaces the rule DPLLB in our DPLL calculus. The notions introduced for DPLL apply to CDCL too, and the rules CDCLU, CDCLD, CDCLE0, and CDCLE1 are identical to their counterparts in the DPLL framework. We therefore concentrate our presentation on rule CDCLJ.

Suppose the current trail is I, and there exists a clause $C \in F$ such that $C|_I = 0$. The idea of CDCL is to *analyze the conflict* and to determine a *conflict clause*[2] D representing the reason of the conflict. By being *learnt*, i. e., added to F, the clause D prevents the solver from repeating this falsifying assignment.[3] Furthermore, it can be used to steer the solver away from the region of the search space containing no solution. This is achieved by backtracking to the *assertion level* al, which is the second greatest decision level of the literals in D. It is also the smallest decision level on I at which D becomes unit: while $D|_I = 0$, we have $D|_J = (\ell)$, where J is the subsequence of I consisting of all literals at decision levels smaller or equal to al and ℓ is the literal in D with the greatest decision level. If the solver would backtrack to a decision level smaller than al, the clause D would not become unit. Backtracking to the assertion level might result in unassigning literals at multiple decision levels, hence it is also called *non-chronological backtracking* or *backjumping*. We sometimes also speak of *conflict-driven backjumping*. To compare CDCL with conflict-driven backjumping with DPLL, we show its working on Example 5.3.

2 not to be confounded with the conflicting clause introduced previously, which is the clause whose literals are all assigned 0 in a conflict

3 as long as D is not deleted from F, as will be explained further down

Table 5.2: CDCL execution for Example 5.4.

STEP	RULE	I	$F\|_I$	LEARNT
0		ε	F	
1	CDCLD	a^d	$(\neg b \vee \neg c) \wedge (b \vee \neg c) \wedge (b \vee c) \wedge (\neg b \vee c)$	
2	CDCLD	$a^d \neg c^d$	$(b) \wedge (\neg b)$	
3	CDCLU	$a^d \neg c^d b^{C_5}$	0	
4	CDCLJ	c^D	$(\neg a \vee \neg b) \wedge (\neg a \vee b)$	$D \overset{\text{def}}{=} (c)$
5	CDCLD	$c^D \neg a^d$	1	$D \overset{\text{def}}{=} (c)$
6	CDCLE1	$c^D \neg a^d$	1	$D \overset{\text{def}}{=} (c)$

Example 5.4 (CDCL search)**.** *Consider again Example 5.3, where the propositional formula $F = C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5 \wedge C_6 = (a \vee b \vee c) \wedge (a \vee \neg b \vee c) \wedge (\neg a \vee \neg b \vee \neg c) \wedge (\neg a \vee b \vee \neg c) \wedge (b \vee c) \wedge (\neg b \vee c)$ is defined over the set of variables $V = \{a, b, c, d\}$. The execution steps are listed in Table 5.2. Steps 1 to 3 are identical to steps 1 to 3 in Example 5.3. Unlike in DPLL, the conflict obtained in step 3 is analyzed. Suppose, conflict analysis returns the conflict clause $D = (c)$. The assertion level of a unit clause is zero per definition, and after adding D to F, the solver jumps back to decision level zero by unassigning all variables except the ones propagated at decision level zero (which in our example did not occur). After backjumping, the unit literal c is propagated at decision level zero with reason D (step 4). The decision $\neg a^d$ is taken (step 5), and F evaluates to 1 under the current assignment. The solver terminates in state SAT (step 6). In total, 6 steps are required to show the satisfiability of F, in contrast to DPLL requiring 10 steps.*

Conflict analysis is an important ingredient in CDCL. In essence, the conflicting clause is resolved with the reason of one of its literals. This procedure is repeated with the reason of one literal in the resolvent, and so on, until the resolvent D contains one single literal at *conflict level*, i. e., the decision level at which the conflict occurred, which in CDCL with non-chronological backtracking is the greatest decision level on the trail I. The resolution mechanism ensures that the clause D is entailed by F. From a practical point of view, this means that conflict clauses can be learnt, i. e., added to F, and that they can also be removed anytime to, e. g., control the size of F. Learning conflict clauses is a prerequisite for conflict-driven backjumping. For this thesis, it is important to remember that the conflict clause D does not only preserve the satisfiability of F but also its models and hence its model count. We will examine the conflict analysis procedure in more detail in Chapter 20, where we adapt it to propositional model enumeration. A more detailed discussion of DPLL and CDCL can be found in Chapters 3 and 4 of the second edition of the Handbook of Satisfiability [58, 126].

6

SAT-BASED EXACT UNWEIGHTED MODEL COUNTING

Let F be a propositional formula defined over the set of variables V. *Propositional model counting*, or #SAT, is the task of determining the number of *total* models of F. In these models, all variables in V are assigned, in contrast to a *partial* model m, in which some variables do not occur. A total assignment obtained by assigning those variables arbitrarily is a *total extension* of m. Every variable can be assigned either 1 or 0, and hence there exist $2^{|V - \mathsf{var}(m)|}$ total extensions of m, where $|V - \mathsf{var}(m)|$ denotes the number of unassigned variables in m. A partial model therefore allows to compactly represent multiple total models, as shown in the following example. We denote with $\mathsf{models}(F)$ and $\mathsf{models}(m)$ the set of all total models of F and the set of all total extensions of m, respectively, and with $\#F$ and $\#m$ their count.

Example 6.1 (Partial models and model count)**.** *Consider again Example 5.1, where $F = (a \wedge c) \vee (b \wedge d)$ and $V = \{a, b, c, d\}$. It has partial models $m_1 = a\,c$ and $m_2 = b\,d$. In both, two variables are unassigned, and m_1 and m_2 represent four total models each. The total models represented by m_1 are $\mathsf{models}(m_1) = \{a\,b\,c\,d, a\,b\,c\,\neg d, a\,\neg b\,c\,d, a\,\neg b\,c\,\neg d\}$, and the ones represented by m_2 are $\mathsf{models}(m_2) = \{a\,b\,c\,d, a\,b\,\neg c\,d, \neg a\,b\,c\,d, \neg a\,b\,\neg c\,d\}$. The model $a\,b\,c\,d$ is represented by both m_1 and m_2, and $\#F = 7$.*

As mentioned in Chapter 5, SAT solvers usually work on formulae in CNF. The same applies to most #SAT solvers, and we are interested in CNF transformation methods which preserve the model count. It turns out that for the Tseitin transformation this is the case, as argued by means of the following example.

Example 6.2 (Model count of Tseitin transformation)**.** *Consider again the formula $F = (a \wedge c) \vee (b \wedge d)$ defined over the set of variables $V = \{a, b, c, d\}$ shown in Example 5.1 and its Tseitin transformation $\mathsf{tseitin}(F) = (\neg t_1 \vee a) \wedge (\neg t_1 \vee c) \wedge (t_1 \vee \neg a \vee \neg c) \wedge (\neg t_2 \vee b) \wedge (\neg t_2 \vee d) \wedge (t_2 \vee \neg b \vee \neg d) \wedge (\neg t_3 \vee t_1 \vee t_2) \wedge (t_3 \vee \neg t_1) \wedge (t_3 \vee \neg t_2)$ with Tseitin variables $T = \{t_1, t_2, t_3\}$ introduced in Example 5.2. We saw that the assignment $\sigma = \{a \mapsto 1, b \mapsto 0, c \mapsto 1, d \mapsto 1\}$ satisfies F. To extend it to an assignment τ satisfying $\mathsf{tseitin}(F)$, we set the Tseitin variables t_1, t_2, and t_3 to the values matching their definition, i. e., $t_1 \leftrightarrow a \wedge c$, $t_2 \leftrightarrow b \wedge d$, and $t_3 \leftrightarrow t_1 \vee t_2$. The resulting assignment is $\tau = \{a \mapsto 1, b \mapsto 0, c \mapsto 1, d \mapsto 1, t_1 \mapsto 1, t_2 \mapsto 0, t_3 \mapsto 1\}$, which is a model of $\mathsf{tseitin}(F)$. The values of the Tseitin variables are uniquely determined by their definition. Therefore, τ is the only total extension of σ satisfying $\mathsf{tseitin}(F)$, and whenever the value of a Tseitin variable does not match its definition, the resulting assignment falsifies $\mathsf{tseitin}(F)$. Hence, the Tseitin transformation preserves the model count of F.*

Most state-of-the-art #SAT solvers implement a component-based approach. The idea is to split the formula under consideration into subformulae, called *components*, with pairwise disjoint variable sets. These subformulae are then processed independently and their model counts multiplied to yield the model count of the original formula. The following example shows the working of this method.

Example 6.3 (Component-based model counting)**.** *Suppose the task is to count the models of the CNF formula $F = C_1 \wedge C_2 \wedge C_3 = (a \vee b) \wedge (a \vee c) \wedge (d \vee e)$ over the set of variables $V = \{a, b, c, d, e\}$. Now, $\mathsf{var}(C_1 \wedge C_2) = \{a, b, c\}$ and $\mathsf{var}(C_2) = \{d, e\}$, and in particular $\mathsf{var}(C_1 \wedge C_2) \cap \mathsf{var}(C_2) = \emptyset$, hence F can be partitioned into two components $\mathcal{C}_1 = C_1 \wedge C_2$ and $\mathcal{C}_2 = C_3$, and $F = \mathcal{C}_1 \wedge \mathcal{C}_2$. The set of variables over which a component is defined, is restricted to the variables occurring in its clauses, i. e., we have $\mathsf{var}(\mathcal{C}_1) = \{a, b, c\}$ and $\mathsf{var}(\mathcal{C}_2) = \{d, e\}$. The models of $\mathcal{C}_1$ are a and $\neg a\, b\, c$, and the model count of $\mathcal{C}_1$ is $\#\mathcal{C}_1 = 5$. The models of $\mathcal{C}_2$ are d and $\neg d\, e$, and $\#\mathcal{C}_2 = 3$. The models of F are $a\, e$, $\neg a\, b\, c\, e$, $a\, d\, \neg e$, and $\neg a\, b\, c\, d\, \neg e$, and $\#F = 15 = \#\mathcal{C}_1 \cdot \#\mathcal{C}_2$.*

Each component might be defined over a potentially small subset of the original set of variables, and the search space to be processed in each computation might be significantly reduced. In addition, if a component is determined to be unsatisfiable, the original formula is unsatisfiable as well, and the model counter can immediately return model count zero [15]. This approach was proposed by Bayardo and Pehoushek [15] as an extension of the DPLL-based model counting method introduced by Birnbaum and Lozinskii [27]. Improvements focused on data structures, such as component caches [12, 172, 179, 189], and algorithms, e. g., for unit propagation and clause learning [173, 189]. Since the components are processed separately, this method is suited for parallel and distributed programming [37, 38].

In Example 6.3, we did not touch upon how the model count of a component is computed. Considering the success of CDCL with conflict-directed backjumping in SAT solving, it would be obvious to use it as a basis for SAT-based model counting. Only minor modifications seem necessary: whenever a model has been found, the most recent decision on the trail is flipped and the search continued until no decisions are left on the trail, similarly to the approach proposed by Birnbaum and Lozinskii [27]. However, CDCL-based model counters might find models multiple times, as is demonstrated by a simple example.

Example 6.4 (Multiple model counts in CDCL-based #SAT solvers)**.** *Consider the formula $F = C_1 \wedge C_2 \wedge C_3 = (\neg a \vee b) \wedge (c \vee d) \wedge (c \vee \neg d)$, which is defined over the set of variables $V = \{a, b, c, d\}$, and let M be a variable keeping track of the number of models found and initialized with zero. Assume our CDCL-based model counter first decides a, followed by propagating b with reason C_1 and deciding c. The resulting trail $I = a^d\, b^{C_1}\, c^d$ satisfies F. It represents two total models, since the variable d does not occur in it, and $M = 2$. The most recent decision, c^d, is flipped. The trail now is $I = a^d\, b^{C_1}\, \neg c$, and $F|_I = (d) \wedge (\neg d)$ contains two unit clauses. After propagating d with reason C_2, a conflict is obtained, since C_3 becomes empty. The conflict is analyzed, i. e., the conflicting clause C_3 is resolved with the reason of d, C_2, obtaining $D = (c)$, which is added to F. Since D is unit, the assertion level is zero, and the #SAT solver jumps to decision level zero, at which no literals were propagated. All assignments are undone, and the unit literal c is propagated with reason D. The trail now is $I = c^D$, and $F|_I = (\neg a \vee b)$ contains no unit literal. After deciding again a followed by propagating b with reason C_1, we have $I = c^D a^d\, b^{C_1}$, which is the partial model found earlier. The model count M is increased by two, and the model count returned eventually will be erroneous.*

This problem does not occur in DPLL-based #SAT solvers, since backtracking explores the search space in a depth-first manner both when finding conflicts and identifying models. The detected (counter-)models are pairwise contradicting, and

$$
\begin{array}{ll}
\text{\#DPLLU:} & (F,\, I,\, M) \rightsquigarrow_{\text{\#DPLLU}} (F,\, I\ell^{C},\, M) \;\text{ if }\; \text{exists } C \in F \text{ with } \; (\ell) = C|_I \\[2ex]
\text{\#DPLLD:} & (F,\, I,\, M) \rightsquigarrow_{\text{\#DPLLD}} (F,\, I\ell^{d},\, M) \;\text{ if }\; F|_I \neq 0 \;\text{ and} \\
 & \text{units}(F|_I) = \varnothing \;\text{ and }\; \text{var}(\ell) \in V - I \\[2ex]
\text{\#DPLLB0:} & (F,\, I,\, M) \rightsquigarrow_{\text{\#DPLLB0}} (F,\, J\bar{\ell},\, M) \;\text{ if }\; \text{exists } C \in F \;\text{ with} \\
 & J\ell^{d}K \overset{\text{def}}{=} I \;\text{ and }\; C|_I = 0 \;\text{ and }\; \text{decs}(K) = \varnothing \\[2ex]
\text{\#DPLLB1:} & (F,\, I,\, M) \rightsquigarrow_{\text{\#DPLLB1}} (F,\, J\bar{\ell},\, M + 2^{|V-I|}) \;\text{ if }\; F|_I = 1 \;\text{ and} \\
 & J\ell^{d}K \overset{\text{def}}{=} I \;\text{ and }\; \text{decs}(K) = \varnothing \\[2ex]
\text{\#DPLLE0:} & (F,\, I,\, M) \rightsquigarrow_{\text{\#DPLLE0}} M \;\text{ if }\; \text{exists } C \in F \;\text{ with }\; C|_I = 0 \;\text{ and} \\
 & \text{decs}(I) = \varnothing \\[2ex]
\text{\#DPLLE1:} & (F,\, I,\, M) \rightsquigarrow_{\text{\#DPLLE1}} M + 2^{|V-I|} \;\text{ if }\; F|_I = 1 \;\text{ and }\; \text{decs}(I) = \varnothing
\end{array}
$$

Figure 6.1: #DPLL rules.

the (partial) models identified during the search represent pairwise disjoint sets of total models. Therefore, most component-based #SAT solvers are based on DPLL.

The fact that chronological backtracking prevents finding models multiple times is crucial in this thesis, obviously for model counting but also for model enumeration. We therefore present a formalization of the core of the DPLL-based model counting method introduced by Birnbaum and Lozinskii [27]. Unlike their algorithm, which is defined recursively, we present an iterative version based on our DPLL calculus shown in Figure 5.1. Non-terminal states are represented by tuples (F, I, M), where F and I represent a propositional formula in CNF defined over a set of variables V and a trail with variables in V, respectively. The third element, M, is an integer denoting the number of models found so far. The initial state is $(F, \varepsilon, 0)$, where ε denotes the empty trail. The end state is M. The transition relation $\rightsquigarrow_{\text{\#DPLL}}$ is the union of transition relations $\rightsquigarrow_{\text{R}}$, where R is either #DPLLU, #DPLLD, #DPLLB0, #DPLLB1, #DPLLE0, or #DPLLE1. The rules of our #DPLL calculus are listed in Figure 6.1. They capture unit propagation (suffix U), decisions (D), backtracking after a conflict (B0) and after a model (B1) and termination with a conflict (E0) and with a model (E1).

The rules #DPLLU, #DPLLD, #DPLLB0, and #DPLLE0 differ from their counterparts in the DPLL framework only in M. Contrarily to DPLLE1, rule #DPLLE1 is applicable only if the trail I contains no decision literal, indicating that all assignments have been checked. Otherwise, the model counter backtracks chronologically (rule #DPLLB1). Whenever the trail I is a (partial) model of F, the number of total models represented by I is added to M, otherwise M remains unaltered. The working of our #DPLL framework is shown by an example.

Example 6.5 (DPLL-based model counting)**.** *Consider again Example 6.3, where* $F = C_1 \wedge C_2 \wedge C_3 = (a \vee b) \wedge (a \vee c) \wedge (d \vee e)$ *and* $V = \{a, b, c, d, e\}$*. The execution steps are listed in Table 6.1. After deciding* a *(step 1) and* d *(step 2), a first model* $m_1 = ad$

Table 6.1: #DPLL execution for Example 6.5.

STEP	RULE	I	$F\|_I$	M
0		ε	$(a \vee b) \wedge (a \vee c) \wedge (d \vee e)$	0
1	#DPLLD	a^d	$(d \vee e)$	0
2	#DPLLD	$a^d\, d^d$	1	0
3	#DPLLB1	$a^d \neg d$	(e)	8
4	#DPLLU	$a^d \neg d\, e^{C_3}$	1	8
5	#DPLLB1	$\neg a$	$(b) \wedge (c) \wedge (d \vee e)$	12
6	#DPLLU	$\neg a\, b^{C_1}$	$(c) \wedge (d \vee e)$	12
7	#DPLLU	$\neg a\, b^{C_1} c^{C_2}$	$(d \vee e)$	12
8	#DPLLD	$\neg a\, b^{C_1} c^{C_2} d^d$	1	12
9	#DPLLB1	$\neg a\, b^{C_1} c^{C_2} \neg d$	(e)	14
10	#DPLLU	$\neg a\, b^{C_1} c^{C_2} \neg d\, e^{C_3}$	1	14
11	#DPLLE1	$\neg a\, b^{C_1} c^{C_2} \neg d\, e^{C_3}$	1	15

of F is found. It contains two out of five variables and therefore represents $2^3 = 8$ total models of F. The model count M is updated, and the model counter backtracks chronologically (step 3). The most recent decision literal, d^d, is flipped, upon which e can be propagated with reason C_3 (step 4). A second model $m_2 = a \neg d e$ is obtained, which represents 4 total models. The models m_1 and m_2 contain contradicting literals, namely d and $\neg d$. Therefore, their total extensions are pairwise contradicting, and m_1 and m_2 represent disjoint sets of total models of F, and we can sum up their model counts. Chronological backtracking occurs, and M is updated accordingly (step 5). The current trail is $I = \neg a$, and $F|_I$ contains two unit literals, b and c. Assume we first propagate b with reason C_1 (step 6) and then c with reason C_2 (step 7). The residual of F under $I = \neg a\, b^{C_1} c^{C_2}$ contains no unit and no empty clause, and there are unassigned variables. Deciding d results in finding a third model $m_3 = \neg a\, b^{C_1} c^{C_2} d^d$ (step 8). Since one variable is unassigned, m_3 represents two total models of F, and M is updated. Chronological backtracking occurs obtaining $I = \neg a\, b^{C_1} c^{C_2} \neg d$ and $F|_I = (e)$ (step 9). The unit literal e is propagated with reason C_3, and a fourth model $m_4 = \neg a b c \neg d e$ is found, which is total. The trail I contains only propagated literals and therefore represents one total model of F. The model count M is incremented by one, and the computation terminates with $M = 15$ (step 11), which coincides with the model count computed in Example 6.3. The model count of F equals the sum of the number of models represented by the detected (partial) models, i. e., $\#F = \#m_1 + \#m_2 + \#m_3 + \#m_4 = 2^3 + 2^2 + 2^1 + 2^0 = 8 + 4 + 2 + 1 = 15$, which equals the model count computed in Example 6.3.

In this chapter, we focused on SAT-based model counting. Unlike in SAT, where the search terminates after finding one model, in #SAT the search space need be processed exhaustively, and #SAT is therefore harder than SAT. Chapter 7 is dedicated to model enumeration without repetitions and introduces *blocking clauses*, which avoid finding models multiple times. Furthermore, we sometimes have to take into account variables, which are not relevant in a given task. The models need be *projected* onto the relevant variables, and in this context we speak of *projected model counting* and *projected model enumeration* addressed in Chapter 8.

7

SAT-BASED IRREDUNDANT MODEL ENUMERATION

Let V be a set of propositional variables and F a (propositional) formula defined over V. *Propositional model enumeration (All-SAT)* is the task of enumerating all models of F. We distinguish between *redundant* model enumeration, where models are allowed to be enumerated multiple times, and *irredundant* model enumeration, in which repetitions must be avoided. The focus in this thesis—and in this chapter—is on irredundant model enumeration.[1] Just like in #SAT, the search space need be processed exhaustively, and in irredundant model enumeration we face the same challenges as far as model detection is concerned.

In Example 6.4 we have seen that model counters based on CDCL with conflict-directed backjumping might detect models multiple times. This behavior can be prevented by the usage of so-called *blocking clauses*. Just as the conflict clause learned during conflict analysis keeps the solver from repeating a bad assignment,[2] a blocking clause ensures that a model is not found again. A simple way to rule out a model is to add its negation to the formula [104, 131, 144].

Example 7.1 (Blocking clause)**.** *Consider again Example 6.4, where $V = \{a, b, c, d\}$ is a set of propositional variables, and $F = C_1 \wedge C_2 \wedge C_3 = (\neg a \vee b) \wedge (c \vee d) \wedge (c \vee \neg d)$ is a propositional formula defined over V. Suppose the model $m = a\,b\,c\,d$ has been found. The corresponding assignment and blocking clause are $\sigma = \{a \mapsto 1, b \mapsto 1, c \mapsto 1, d \mapsto 1\}$, and $B = \neg m = (\neg a \vee \neg b \vee \neg c \vee \neg d)$, respectively, and B is added to F. Suppose later in the search the assignment $\tau = \{a \mapsto 1, b \mapsto 1, c \mapsto 1\}$ is encountered. Since $B|_\tau = (\neg d)$, the literal $\neg d$ is propagated with reason B obtaining $v = \{a \mapsto 1, b \mapsto 1, c \mapsto 1, d \mapsto 0\}$, which differs from σ. Hence, any model of F matching σ will not be found again.*

As soon as all models are found and the corresponding blocking clauses are added, the formula F becomes unsatisfiable. A very basic All-SAT solver could therefore execute CDCL with non-chronological backjumping as in SAT solving. Whenever a model has been found and not all assignments have been tested, it would add the corresponding blocking clause to F, undo all assignments at decision levels higher than zero, and continue its search. If it finds no model at all, the input formula is unsatisfiable, and the solver terminates in state $M = 0$.

To make this idea more concrete, we extend our CDCL framework presented in Figure 5.2 accordingly. Non-terminal states are represented by tuples (F, I, M), where F denotes a formula defined over the set of variables V, and I denotes a trail with variables in V. The third element M is a propositional formula in *disjunctive normal form (DNF)*, which is a disjunction of *cubes*, which are conjunctions of literals. The initial state is $(F, \varepsilon, 0)$, where 0 denotes the empty DNF formula. The end

1 In Chapter 20, we additionally present a method for enumerating redundant short partial models.
2 as long as this conflict clause is not deleted

AllCDCLU: $(F, I, M) \rightsquigarrow_{\mathsf{AllCDCLU}} (F, I\ell^C, M)$ if exists $C \in F$ with $(\ell) = C|_I$

AllCDCLD: $(F, I, M) \rightsquigarrow_{\mathsf{AllCDCLD}} (F, I\ell^d, M)$ if $F|_I \neq 0$ and $\mathsf{units}(F|_I) = \emptyset$ and $\mathsf{var}(\ell) \in V - I$

AllCDCLJ: $(F, I, M) \rightsquigarrow_{\mathsf{AllCDCLJ}} (F \wedge D, J\ell^D, M)$ if exists $C \in F$ with $JK \overset{\text{def}}{=} I$ and $C|_I = D|_I = 0$ and $(\ell) = D|_J$ and $F \models D$

AllCDCLB1: $(F, I, M) \rightsquigarrow_{\mathsf{AllCDCLB1}} (F \wedge B, J\overline{\ell}, M \vee I)$ if $F|_I = 1$ and $J\ell^d K \overset{\text{def}}{=} I$ and $B \overset{\text{def}}{=} \neg I$ and $\mathsf{decs}(K) = \emptyset$

AllCDCLE0: $(F, I, M) \rightsquigarrow_{\mathsf{AllCDCLE0}} M$ if exists $C \in F$ with $C|_I = 0$ and $\mathsf{decs}(I) \neq \emptyset$

AllCDCLE1: $(F, I, M) \rightsquigarrow_{\mathsf{AllCDCLE1}} M \vee I$ if $F|_I = 1$ and $\mathsf{decs}(I) = \emptyset$

Figure 7.1: All-CDCL rules.

state is M whose cubes are the models of F, if F is satisfiable, and $M = 0$ otherwise, and which therefore is logically equivalent to F. The transition relation $\rightsquigarrow_{\mathsf{AllCDCL}}$ is the union of the transition relations $\rightsquigarrow_{\mathsf{R}}$, where R is either AllCDCLU, AllCDCLD, AllCDCLJ, AllCDCLB1, AllCDCLE0, or AllCDCLE1.

The rules of our All-CDCL framework are depicted in Figure 7.1. Analogously to our CDCL calculus, they describe unit propagation (suffix U), decisions (D), conflict-driven backjumping (J), and backtracking after a model (B1) as well as termination with a conflict (E0) or a model (E1). The rules AllCDCLU, AllCDCLD, AllCDCLJ, and AllCDCLE0 differ from their counterpart in the CDCL framework in Figure 5.2 only in the additional element M in non-terminal states and in the end state. The end rule AllCDCLE1 is applicable only if no decision is left on the trail I. The rule AllCDCLB1 corresponds to the rule #DPLLB1 in our #DPLL framework in Figure 6.1 but with adding a blocking clause to F and recording the satisfying assignment instead of updating the model count. We show the working of our calculus by means of a small example.

Example 7.2 (CDCL-based model enumeration with blocking clauses)**.** *Consider the formula $F = C_1 \wedge C_2 = (a \vee b) \wedge (\neg a \vee b)$ defined over the set of variables $V = \{a, b\}$. The execution steps are listed in Table 7.1. After deciding a (step 1) and propagating b with reason C_2, a first model ab is found (step 2). The corresponding blocking clause is $B = (\neg a \vee \neg b)$. It is added to F resulting in $F = (a \vee b) \wedge (\neg a \vee b) \wedge (\neg a \vee \neg b)$, and the model is added to M, which gives us $M = (a \wedge b)$. No assignments occurred at decision level zero, and all assignments are undone (step 3). We decide again a, and this time $F|_I$ contains two unit clauses, namely $C_2|_I = (b)$ and $B|_I = (\neg b)$ (step 4). We choose to propagate b with reason C_2, upon which the blocking clause B becomes empty (step 5). Although the current assignment ab is a model of F, it is blocked by B and is not enumerated again. The conflict is analyzed by resolving the conflicting clause B with the reason of b, C_2, and the unit clause $D = (\neg a)$ is learnt. Our formula is now*

Table 7.1: All-CDCL execution for Example 7.2.

STEP	RULE	I	$F\|_I$	M	LEARNT
0		ε	$(a \vee b) \wedge (\neg a \vee b)$	0	
1	AllCDCLD	a^d	(b)	0	
2	AllCDCLU	$a^d\, c^{C_2}$	1	0	
3	AllCDCLB1	ε	$(a \vee b) \wedge (\neg a \vee b) \wedge (\neg a \vee \neg b)$	$(a \wedge b)$	
4	AllCDCLD	a^d	$(b) \wedge (\neg b)$	$(a \wedge b)$	
5	AllCDCLU	$a^d\, b^{C_2}$	0	$(a \wedge b)$	
6	AllCDCLU	$\neg a^D$	(b)	$(a \wedge b)$	$D \overset{\text{def}}{=} (\neg a)$
7	AllCDCLU	$\neg a^D\, b^{C_1}$	1	$(a \wedge b)$	$D \overset{\text{def}}{=} (\neg a)$
8	AllCDCLE1	$\neg a^D\, b^{C_1}$	1	$(a \wedge b) \vee (\neg a \wedge b)$	$D \overset{\text{def}}{=} (\neg a)$

$F = (a \vee b) \wedge (\neg a \vee b) \wedge (\neg a \vee \neg b) \wedge (\neg a)$. Learning a unit forces the model enumerator to jump to decision level zero and to propagate this unit literal (step 6). We now have $F|_I = (b)$, and b is propagated with reason C_1 (step 7). A second model $\neg a\, b$ has been found. It is added to M, and since I contains no decisions, the computation terminates in state $M = (a \wedge b) \vee (\neg a \wedge b)$ (step 8). Notice that $a\,b$ is a model of the original formula F, but not of $F \wedge B \wedge D$, whereas $\neg a\, b$ is a model of F and also of $F \wedge B \wedge D$.

Clearly, blocking clauses must not be deleted anytime, and in the worst case their number is exponential in the number of variables $|V|$. Consequently, the formula F might increase exponentially in size. Furthermore, blocking clauses might contain a great number of literals, and long clauses do not propagate easily. The obvious consequence is a potential negative impact on solver performance and memory requirements. We are therefore interested in finding short blocking clauses and adding the fewest possible of them to F.

To obtain shorter blocking clauses, Morgado and Marques-Silva [144] propose to consider only the negated decision literals, i. e., to define $B = \neg\text{decs}(I)$, where I is a trail satisfying F. However, this does not affect their number, since the values of the unconsidered (propagated) literals are uniquely determined. The number and the size of blocking clauses can also be reduced by determining short partial models. The shorter a model, the higher the number of total models it represents and the larger the portion of the search space ruled out by the corresponding blocking clause. In this context, we also speak of *search space pruning*. Considerable effort has therefore been put into devising methods for *shrinking*, or shortening, (total) models. McMillan [131] proposed to execute conflict analysis on the input formula represented as a circuit and a total satisfying assignment. A similar method was introduced by Jin, Han, and Somenzi [103]. After finding a model, unit propagation is executed on the original formula represented as a circuit and the just detected satisfying assignment until the circuit is satisfied. The negation of this satisfying assignment is added to F as a blocking clause, which becomes empty under this satisfying assignment. The assignments involved in the conflict are identified by means of conflict analysis, which in turn might give a shorter model. A different approach was taken by Gebser, Kaufmann, and Schaub [84]. Although designed for Answer Set Programming (ASP), their method can readily be adapted to All-SAT. In essence, they designed the decision strategy such that blocking clauses become obsolete later in the search and can safely be removed from F. This ensures that anytime F contains at most a number of blocking clauses which is polynomial in the number of variables $|V|$.

Blocking clauses not only ensure that models are not enumerated multiple times, they also provide the reasons of flipped decision literals. This is essential in CDCL, since conflict analysis presupposes that for every literal on the trail which is not a decision, its reason is contained in F. Assume a blocking clause B consists of the flipped decision literals on a satisfying trail. If all but the most recent decision in B are repeated, the blocking clause B becomes unit, as in Example 7.1, and the negation of this decision is propagated. In this sense, the clause B acts as reason for the most recent decision flipped after finding a model: it becomes unit after the assignments on the greatest decision level are undone, and its unit literal is the negation of the most recent decision. A simple example clarifies this idea.

Example 7.3 (Reason of flipped decision)**.** *Consider again the situation in Example 7.1 where $F = C_1 \wedge C_2 \wedge C_3 = (\neg a \vee b) \wedge (c \vee d) \wedge (c \vee \neg d)$ and $V = \{a, b, c, d\}$, and the model $m = abc$ of F has just been found. Suppose the trail is $I = a^d\, b^{C_1}\, c^d$. If we only add the negated decisions to the blocking clause, we obtain $B = (\neg a \vee \neg c)$. Flipping the most recent decision literal involves undoing all assignments starting from the most recent decision. In our example, only c is affected, and the resulting trail is $J = a^d\, b^{C_1}$. Now B becomes unit, i. e., $B|_J = (\neg c)$, and $(F \wedge B)|_J = (c \vee d) \wedge (c \vee \neg d) \wedge (\neg c)$. The literal $\neg c$ is propagated with reason B, and the resulting trail is $K = a^d\, b^{C_1}\, \neg c^B$, which is exactly I after flipping the most recent decision literal c.*

Example 7.3 shows that without the use of blocking clauses the model enumerator might face problems during conflict analysis. Grumberg, Schuster, and Yadgar [94] address this issue by introducing sub-levels for flipped decisions. During conflict analysis, flipped decisions are treated similarly to decisions. Toda and Soh [191] take a different approach by modifying the conflict analysis scheme.

Repetitions can also be prevented by using exclusively chronological backtracking as in #DPLL shown in Figure 6.1 but by storing the models instead of the model count. In this manner, all decisions are flipped in reverse assignment order, and each assignment occurs exactly once, which renders the use of blocking clauses obsolete. However, as shown above, DPLL-based model enumerators are not able to escape search space regions without model early, which is a significant drawback compared to CDCL with conflict-driven backjumping. Therefore, the challenge in propositional model enumeration is twofold: on the one hand we want to avoid (or at least limit) the use of blocking clauses, and on the other hand we want to take advantage of conflict analysis and conflict-directed backjumping.

After focusing on SAT-based model counting and enumeration, we now turn our attention to *projection*. In some tasks, only a subset of the variables is relevant. This poses particular challenges, since the models need be projected onto these relevant variables. Common solutions are presented in Chapter 8.

8

PROJECTION

Let X and Y be two disjoint sets of propositional variables, and let $F(X,Y)$ be a propositional formula defined over the set of variables $X \cup Y$. If for our application only the variables in X are relevant, we are interested in the models of F *projected* onto X. Stated otherwise, we *existentially quantify* F over Y and write $\exists Y \,.\, F(X,Y)$. The models of $\exists Y \,.\, F(X,Y)$ are the models of F projected onto X, and projected model counting is accordingly also referred to as #∃SAT. In this sense, #SAT can be seen as a special case of #∃SAT, namely the one in which $Y = \emptyset$ [202].

Example 8.1 (Projected model counting and enumeration)**.** *Consider again Example 5.1 where* $F = (a \wedge c) \vee (b \wedge d)$*. The total models of* F *are* $\mathsf{models}(F) = \{abcd,$ $abc\neg d,\ a\neg bcd,\ a\neg bc\neg d,\ ab\neg cd,\ \neg abcd,\ \neg ab\neg cd\}$ *and* $\#F = 7$ *(see Example 6.1). Now assume* $X = \{a,b\}$ *and* $Y = \{c,d\}$*. The total models of* F *projected onto* X *are* $\mathsf{models}(\exists Y \,.\, F(X,Y)) = \{ab, a\neg b, \neg ab\}$*, and* $\#(\exists Y \,.\, F) = 3$*.*

Instead of existentially quantifying F over Y, in practice one would remove from the models of F all literals with variable in Y. However, Example 8.1 shows that it is not sufficient to just project the models of $F(X,Y)$ onto X: two models differing only in literals with variable in Y are considered the same model projected onto X and hence only count as one model. This is the case, e. g., for $abcd$ and $ab\neg cd$. One must therefore prevent the detection of multiple models differing only in literals with irrelevant variables. This can be achieved either by using blocking clauses or by prioritizing decisions of relevant variables over decisions of irrelevant variables and by backtracking chronologically over relevant decisions and non-chronologically over irrelevant decisions [84, 94].

Projection plays an important role in counting the models of the CNF transformation of an arbitrary formula $F(V)$. As argued in Chapter 6, the Tseitin transformation of F, denoted $\mathsf{tseitin}(F)$, preserves the model count of F. Furthermore, the models of $\mathsf{tseitin}(F)$ projected onto V, resp. the models of $\exists T \,.\, \mathsf{tseitin}(F)$, where T denotes the Tseitin variables, are exactly the models of F.

Example 8.2 (Tseitin transformation and projection)**.** *Consider again the Tseitin transformation in Example 5.2. It has 7 total models:* $abcdt_1t_2$, $abc\neg dt_1\neg t_2$, $a\neg bcdt_1\neg t_2$, $a\neg bc\neg dt_1\neg t_2$, $ab\neg cd\neg t_1t_2$, $\neg abcd\neg t_1t_2$, *and* $\neg ab\neg cd\neg t_1t_2\}$*. Their projection onto the relevant variables* X *gives exactly the models of* F *listed in Example 8.1.*

Exact projected model counting occurs, e.g., in planning [10, 202] and product configuration [203], and several #∃SAT solvers exist [10, 116]. Projected model enumeration is applied in automotive configuration [203], existential quantifier elimination [32], image computation [94, 95], predicate abstraction [118], and bounded model checking [184].

Part III

DUAL PROJECTED MODEL COUNTING

9

PAPER 1: AN ABSTRACT DUAL PROPOSITIONAL MODEL COUNTER

PUBLISHED. In: *YSIP2 – Proceedings of the Second Young Scientist's International Workshop on Trends in Information Processing, Dombai, Russian Federation, May 16–20, 2017*. Ed. by Steffen Hölldobler, Andrey Malikov, and Christoph Wernhard. Vol. 1837. CEUR Workshop Proceedings. CEUR-WS.org, 2017, pp. 17–26. URL: http://ceur-ws.org/Vol-1837/paper5.pdf.

AUTHORS. Armin Biere, Steffen Hölldobler, and Sibylle Möhle.[1]

ACKNOWLEDGMENTS. The authors acknowledge support by the Deutsche Forschungsgesellschaft (DFG) under grant HO 1294/11-1 and by the Austrian Science Fund (FWF) project W1255-N23. We also want to thank Andreas Fröhlich helping us in the early brain-storming and discussing phase of this idea.

ABSTRACT. Various real-world problems can be formulated as the task of counting the models of a propositional formula. This problem, also called #SAT, is therefore of practical relevance. We present a formal framework describing a novel approach based on considering the formula in question together with its negation. This method enables us to close search branches earlier. We formalize a non-dual variant and argue that our framework is sound.

9.1 INTRODUCTION

The problem #SAT consists in determining the number of models of a propositional formula. Applications can be found in a variety of real-world domains, such as reasoning [120, 167], model-based diagnosis of physical systems [112], product configuration in the automotive industry [110], planning [162], and frequent itemset mining [96]. The breadth of these applications emphasizes the practical relevance of #SAT.

Birnbaum and Lozinskii presented an algorithm for counting propositional models based on the Davis Putnam Procedure [27]. In [15], the authors extend this method by splitting the formula in question into subformulae over disjoint sets of variables. The model count is then obtained by multiplying the model counts of these subformulae. In Cachet [172] clause learning and component caching are combined. sharpSAT [189] builds upon Cachet introducing a new component caching scheme. Projected model counting was implemented in [10, 104]. Finally, in [37], a parallel approach is implemented.

1 The authors are listed in alphabetical order.

With a similar motivation as [72] but for #SAT instead of QBF in the spirit of [155], we present a formal framework describing a #SAT solving procedure based on DPLL, called *Abstract Dual #DPLL*, and a formalization of a non-dual variant and argue that our framework is sound. The basic idea of our dual approach consists in executing DPLL on a formula as well as on its negation. In our counting algorithm, we follow the main idea presented in [27]. We implement our framework in SWI-Prolog [199] making use of the PIE system [198]. First experiments showed the suitability of our approach. While a dual approach was addressed in QBF [72, 93], we are not aware of any work on #SAT aiming in this direction.

The paper is structured as follows: After giving some background information, in Section 9.3 we present the rules for dual propositional model counting underlying our counting procedure. In Section 9.4 we introduce our framework and argue about its soundness. By means of an example, in Section 9.5 we demonstrate the operation of our framework as well as of a non-dual variant, before in Section 9.6 we conclude and point out future work. Our notation is based on the one introduced in [97].

9.2 preliminaries

9.2.1 *Propositional Satisfiability and Model Counting*

Let V be a fixed finite set of propositional variables. A *literal* ℓ is either a variable a (*positive literal*) or a negated variable $\neg a$ (*negative literal*). We denote with $\mathsf{var}(\ell)$ the variable of ℓ. The *complement* $\overline{\ell}$ of a literal ℓ is its negation, i. e., $\overline{\ell} = \neg a$ if $\ell = a$, and $\overline{\ell} = a$ if $\ell = \neg a$.

A propositional formula F over variables in V is in conjunctive normal form (CNF), if it is a conjunction of clauses. A *clause* is a disjunction of literals. We denote with $\mathsf{var}(F)$ the set of variables occurring in F.

We define an *interpretation* I as a mapping from the set of variables V to the set of truth values $\{1, 0\}$. If $I(a) \in \{1, 0\}$ for all $a \in V$, then I is called a *total interpretation*. Otherwise, I is said to be a *partial interpretation*. An interpretation may be represented by a sequence of literals containing no pair of complementary literals where each literal occurs at most once. An empty sequence is represented by ε. Let $I = \ell_1 \dots \ell_m$ be a sequence of literals representing an interpretation over V. We say that a literal $\ell \in I$ iff $\ell = \ell_k$ for a $k \in \{1, \dots, m\}$. Let $I' = \ell_{m+1} \dots \ell_n$ be another sequence of literals representing an interpretation over V. We define the *concatenation* of I and I' as $I I' = \ell_1 \dots \ell_n$. With $I \ell I' = \ell_1 \dots \ell_m \ell \ell_{m+1} \dots \ell_n$ we denote the concatenation of I, ℓ, and I'. Note that for $I I'$ and $I \ell I'$ to represent interpretations, they have to meet the requirements given above. We interpret a sequence of literals over different variables also as a set of literals as well as the (possibly partial) interpretation which sets all its literals to true and vice versa. In the rest of this paper, I will denote an interpretation assuming an appropriate representation.

An interpretation I satisfies a positive literal ℓ with variable a, in symbols $I \models \ell$, iff $I(a) = 1$. Analogously, I satisfies a negative literal ℓ with variable a iff $I(a) = 0$. Since a clause C is a disjunction of literals, $I \models C$ iff $I \models \ell$ for a literal $\ell \in C$. Analogously, $I \models F$ iff $I \models C$ for all clauses C of a formula F, since F is defined as a conjunction of clauses. Whenever $I \models F$, we say that I is a *model* for F where I can be partial representing a *partial model* or total representing a *total model*. The *model count* $\#F$ of a formula F corresponds to the number of total models of F.

Two formulae F and G are *semantically equivalent*, denoted by $F \equiv G$, iff for all interpretations I the following holds: $I \models F$ iff $I \models G$. Thus, two formulae are semantically equivalent iff they have the same models.

The *reduct* of a formula F with respect to an interpretation I is given by $F|_I = \{C|_I \mid C \in F \text{ and } C \cap I = \emptyset\}$, where $C|_I = \{\ell \mid \ell \in C \text{ and } \overline{\ell} \notin I\}$. Let $F = (x_1) \wedge (x_2 \vee x_3)$ be a formula over $V = \{x_1, x_2, x_3\}$. In set notation, $F = \{\{x_1\}, \{x_2, x_3\}\}$. Let $I = \{x_1, \neg x_2\}$ be an interpretation over V. Then, $F|_I = \{\{x_3\}\}$. Whenever $F|_I = 1$, I is a model of F. We say that I *satisfies* F and may refer to I as a *satisfying interpretation* where adequate. If $0 \in F|_I$, we say that a *conflict* arises in $F|_I$ or that I *falsifies* F and call I a *falsifying interpretation*. In our example, I neither satisfies nor falsifies F.

9.2.2 *The Davis Putnam Logemann Loveland Procedure*

The Davis Putnam Logemann Loveland (DPLL) procedure [60] is based on the Davis Putnam Procedure (DPP) [61] and conducts a systematic search in the space of all possible interpretations. This space can be visualized as a binary search tree where each node represents a partial interpretation and each leaf represents a total interpretation. DPLL can be visualized as a depth-first tree search based mainly on *unit propagation*, *decisions*, and *backtracking*.

Let F be a formula and I an interpretation over V. If during search a *unit clause* $\{\ell\}$ occurs in $F|_I$, the *unit literal* ℓ must be assigned the value 1 to make I satisfy F. This is ensured by unit propagation. ℓ is called *propagation literal*, and we say that ℓ's value is *implied* by $F|_I$. If there is no unit clause in $F|_I$, a *decision literal* ℓ is chosen and assigned a value. If I falsifies F, backtracking occurs, i. e., all assignments up to the latest decision are undone and the value of the decision literal flipped. The search continues with I modified accordingly. For a more detailed description we refer to [57].

9.2.3 *Counting Models by Means of the Davis Putnam Procedure*

Let F be a formula and I an interpretation over variables V. By means of a decision with respect to a literal ℓ, the set of models of F is split into two disjoint sets. In one $I(\ell) = 1$, in the other $I(\ell) = 0$. Hence, $\#F|_I = \#F|_{I \cup \{\ell\}} + \#F|_{I \cup \{\overline{\ell}\}}$. Based on this observation, a method for counting models based on the Davis Putnam procedure [61] was presented in [27]. The corresponding pseudocode is depicted in Algorithm 1.

It is important to note that, unlike in SAT solving, after determining a satisfying interpretation, the search continues until the entire search space has been processed. The model count is built recursively according to the computation discussed in the previous paragraph.

We now present rules to determine $\#F$ based on Algorithm 1. Let P be a CNF formula over V such that $P \equiv F$. $\#F|_I = \#P|_I = 2^m$ with $m = |V| - |I|$ representing the number of unassigned variables. Now $\#P|_I$ can be computed by the following rules: Whenever I falsifies P, $\#P|_I = 0$; whenever I satisfies P, $\#P|_I = 2^{|V|-|I|}$; whenever $\#P|_I$ is undefined, then $\#P|_I = \#P|_{I \cup \{\ell\}} + \#P|_{I \cup \{\overline{\ell}\}}$, where $\mathsf{var}(\ell) \in V$ and $\{\ell, \overline{\ell}\} \cap I = \emptyset$.

```
function CDP($F$, $|V|$)                                  ▷ $F$ formula over a set of variables $V$
    if $F$ is empty then                                                    ▷ $F$ is satisfiable
      return $2^{|V|}$
    else if $F$ contains an empty clause then                             ▷ $F$ is unsatisfiable
      return $0$
    else if $F$ contains a unit clause $\{\ell\}$ then                     ▷ Unit propagation
      return CDP($F|_{\{\ell\}}$, $|V| - 1$)
    else
        choose a variable $a \in V$
      return CDP($F|_{\{a\}}$, $|V| - 1$) + CDP($F|_{\{\neg a\}}$, $|V| - 1$))
    end if
end function
```

Algorithm 1: Counting by Davis-Putnam (CDP) [27].

9.2.4 *An Abstract Framework for Propositional Model Counting*

Let F be a formula over V and P be a CNF formula over V such that $P \equiv F$. We describe *Abstract #DPLL* as a state transition system $(S, \rightsquigarrow)$ with a set of states S and a binary transition relation $\rightsquigarrow \subseteq S \times S$. The set of states is defined by $S := \mathbb{N} \cup \{(P, I, M) \mid P$ is a CNF formula such that $P \equiv F$, I is an interpretation, $M \in \mathbb{N}\}$, and $\rightsquigarrow := \{\rightsquigarrow_{\mathsf{Dec}}, \rightsquigarrow_{\mathsf{NB1}}, \rightsquigarrow_{\mathsf{NB0}}, \rightsquigarrow_{\mathsf{End1}}, \rightsquigarrow_{\mathsf{End0}}\}$.

In this context, P is called *working formula*, I is called *working interpretation*, and M is called *working number of models*. The initial state is defined by $(P, \varepsilon, 0)$. The terminal state is $\#P = \#F$. ℓ^d denotes a *decision literal*, i.e., a literal which was assigned a value by a decision. Analogously, ℓ denotes a *propagation literal*. The rules are presented in Figure 9.1.

By means of the Dec rule, the working interpretation is extended by an unassigned decision literal ℓ^d whose variable occurs in V. Naive backtracking is applied whenever the working interpretation either satisfies or falsifies P and if it contains a decision literal, i. e., not not all possible interpretations have been tested yet. In the former case, the model count has to be incremented by 2^m, where m represents the number of unassigned variables (NB1). In the latter case, the model count remains unchanged (NB1). The procedure terminates when I either satisfies (End1) or falsifies (End0) P and does not contain any decision literal, i.e., all possible interpretations have been tested. The model count is updated accordingly.

If our framework is sound, every implementation which can be modeled by means of it is sound as well. This comprises optimizations, such as unit propagation. Restricting our framework to a minimal set of rules simplifies the presentation since less cases have to be distinguished and reasoned about.

9.3 COUNTING MODELS BY TAKING INTO ACCOUNT THE NEGATED FORMULA

Let F be a formula over variables V, P and N be CNF formulae over V such that $P \equiv F$ and $N \equiv \neg F$, respectively. Then, $\#F = \#P = 2^{|V|} - \#N$. Further let I be an interpretation over V. For the reduct $F|_I$ the same observation holds: $\#F|_I = \#P|_I = 2^{|V|-|I|} - \#N|_I$, where $|V| - |I|$ represents the number of unassigned variables. Given F, P, N, and V, we define the counting algorithm c taking as input I:

Dec: $(P, I, M) \rightsquigarrow_{\mathsf{Dec}} (P, I\ell^d, M)$ iff $\mathsf{var}(\ell) \in V$ and $\{\ell, \overline{\ell}\} \cap I = \emptyset$

NB⊤: $(P, I\ell^d I', M) \rightsquigarrow_{\mathsf{NB\top}} (P, I\overline{\ell}, M + 2^{|V|-|I\ell I'|})$ iff
$P|_{I\ell I'} = 1$ and I' contains only propagation literals

NB0: $(P, I\ell^d I', M) \rightsquigarrow_{\mathsf{NB0}} (P, I\overline{\ell}, M)$ iff
$0 \in P|_{I\ell I'}$ and I' contains only propagation literals

End1: $(P, I, M) \rightsquigarrow_{\mathsf{End1}} M + 2^{|V|-|I|}$ iff
$P|_I = 1$ and I contains only propagation literals

End0: $(P, I, M) \rightsquigarrow_{\mathsf{End0}} M$ iff
$0 \in P|_I$ and I contains only propagation literals

Figure 9.1: Rules in Abstract #DPLL. P is a CNF formula over variables V, I and I' are interpretations over disjoint sets of variables, ℓ is a literal and $M \in \mathbb{N}$. The procedure is based on DPLL combined with naive backtracking and terminates when the entire search space has been processed.

R1:	$\mathsf{c}(I) = 0$	if $0 \in P\|_I$	Conflict in $P\|_I$
R2:	$\mathsf{c}(I) = 2^{\|V\|-\|I\|}$	if $P\|_I = 1$	$I \models P$
R3:	$\mathsf{c}(I) = 2^{\|V\|-\|I\|}$	if $0 \in N\|_I$	Conflict in $N\|_I$
R4:	$\mathsf{c}(I) = 0$	if $N\|_I = 1$	$I \models N$
R5:	$\mathsf{c}(I) = \mathsf{c}(I \cup \{\ell\}) + \mathsf{c}(I \cup \{\overline{\ell}\})$ where $\mathsf{var}(L) \in V$ and $\{\ell, \overline{\ell}\} \cap I = \emptyset$	else	

If a conflict in $P|_I$ arises, I can not be extended to any total model for F, and the model count is 0 (R1). Whenever $I \models P$, I can be extended to $2^{|V|-|I|}$ total models for F, where $|V| - |I|$ is the number of unassigned variables (R2). In case of a conflict in $N|_I$, I is a model for F and can be extended to $2^{|V|-|I|}$ total models for F (R3). Whenever $I \models N$, I can not be extended to a total model for F, and the model count is 0 (R4). If both $P|_I$ and $N|_I$ are undefined, $\#F|_I = \#N|_{I\cup\{\ell\}} + \#N|_{I\cup\{\overline{\ell}\}}$ (R5) [27].

Unit propagation in $P|_I$ can be simulated by rules R5 and R1:

$$\mathsf{c}(I) = \mathsf{c}(I \cup \{\ell\}) + \underbrace{\mathsf{c}(I \cup \{\overline{\ell}\})}_{0} = \mathsf{c}(I \cup \{\ell\}) \quad \text{if} \quad \{\ell\} \in P|_I \tag{9.1}$$

$\{\ell\}$ is a unit clause in $P|_I$, and therefore ℓ must be set to 1 for $I \cup \{\ell\}$ to satisfy P. This implies that $I \cup \{\overline{L}\}$ falsifies P, i.e., $\mathsf{c}(I \cup \{\overline{\ell}\}) = 0$ according to rule R1.

Unit propagation in $N|_I$ can be simulated by rules R5 and R3:

$$\mathsf{c}(I) = \mathsf{c}(I \cup \{\ell\}) + \underbrace{\mathsf{c}(I \cup \{\bar{\ell}\})}_{2^{|V|-|I\cup\{\bar{\ell}\}|}} = \mathsf{c}(I \cup \{\ell\}) + 2^{|V|-|I\cup\{\bar{\ell}\}|} \quad \text{if} \quad \{\ell\} \in N|_I \tag{9.2}$$

$\{\ell\}$ is a unit clause in $N|_I$, and therefore ℓ must be set to 1 for $I \cup \{\ell\}$ to satisfy N. This implies that $I \cup \{\bar{\ell}\}$ falsifies N, i.e., $I \cup \{\bar{\ell}\}$ satisfies F and can be extended to $2^{|V|-|I\cup\{\bar{\ell}\}|}$ total models for F according to rule R3.

9.4 ABSTRACT DUAL #DPLL

Let F be a formula over variables V, P and N be CNF formulae over V such that $P \equiv F$ and $N \equiv \neg F$, respectively. Note that in particular $\mathsf{var}(F) = \mathsf{var}(P) = \mathsf{var}(N) = V$. Let I be an interpretation. Clearly, $I \models P$ iff $I \not\models N$ and vice versa. P and N are passed to a DPLL [60] solver which works on both formulae simultaneously. The model count is computed according to the rules introduced in Section 9.3.

If a conflict in $P|_I$ arises, I can not be extended to a total model for F, and the model count is 0. Whenever $P|_I = 1$, I satisfies P and can be extended to 2^m total models for F, where m is the number of unassigned variables. This model count is added up to the number of models found so far. In both cases, the solver backtracks chronologically and flips the value of the decision literal turning it into a propagation literal. The search terminates if I contains no decision literal, indicating that the whole search space has been processed.

If a conflict arises in $N|_I$, I is a model for F and can be extended to 2^m total models for F, where m is the number of unassigned variables. This model count is added up to the number of models found so far. Whenever $N|_I = 1$, I satisfies N and can not satisfy F. In both cases, the solver backtracks chronologically and flips the value of the decision literal turning it into a propagation literal. The search terminates if I does not contain any decision literal, indicating that the whole search space has been processed.

Whenever both $P|_I$ and $N|_I$ are undefined, an unassigned literal ℓ is chosen and assigned a value becoming a decision literal. The search continues with I extended by ℓ.

In our version of DPLL, every node is visited at most once, and there is no need to remember the models found so far, e. g., by adding blocking clauses (i. e., the negated models) to P to avoid finding models several times. This ensures that the model count returned by the algorithm corresponds to the number of models of F.

9.4.1 *States and Transition Relations*

Based on Abstract #DPLL introduced in Section 9.2.4, we describe *Abstract Dual #DPLL* as a state transition system $(S, \rightsquigarrow)$ with set of states S and transition relation $\rightsquigarrow \subseteq S \times S$ as follows:

$$\begin{aligned} S &:= \mathbb{N} \cup \{(P, N, I, M) \mid P, N \text{ are CNF formulae with} \\ &\qquad P \equiv F \text{ and } N \equiv \neg F, I \text{ is an interpretation}, M \in \mathbb{N}\} \\ \rightsquigarrow &:= \{\rightsquigarrow_{\mathsf{Dec}}, \rightsquigarrow_{\mathsf{NB1}}, \rightsquigarrow_{\mathsf{NB0}}, \rightsquigarrow_{\mathsf{End1}}, \rightsquigarrow_{\mathsf{End0}}\} \end{aligned}$$

In this context, P and N are called *working formulae*, I is called *working interpretation*, and M is called *working model count*. The initial state is defined by $(P, N, \varepsilon, 0)$. The

$$\begin{aligned}
&\mathsf{Dec}: && (P,\, N,\, I,\, M) \leadsto_{\mathsf{Dec}} (P,\, N,\, I\ell^d,\, M) \quad \text{iff} \\
& && \mathsf{var}(\ell) \in V \quad \text{and} \quad \{\ell, \bar{\ell}\} \cap I = \emptyset \\[1em]
&\mathsf{NB1}: && (P,\, N,\, I\ell^d I',\, M) \leadsto_{\mathsf{NB1}} (P,\, N,\, I\bar{\ell},\, M + 2^{|V|-|I\ell I'|}) \quad \text{iff} \\
& && (0 \in N|_{I\ell I'} \;\text{ or }\; P|_{=}1) \quad \text{and} \quad I' \text{ contains only propagation literals} \\[1em]
&\mathsf{NB0}: && (P,\, N,\, I\ell^d I',\, M) \leadsto_{\mathsf{NB0}} (P,\, N,\, I\bar{\ell},\, M) \quad \text{iff} \\
& && (0 \in P|_{I\ell I'} \;\text{ or }\; N|_{I\ell I'} = \emptyset) \quad \text{and} \quad I' \text{ contains only propagation literals} \\[1em]
&\mathsf{End1}: && (P,\, N,\, I,\, M) \leadsto_{\mathsf{End1}} M + 2^{|V|-|I|} \quad \text{iff} \\
& && (0 \in N|_I \;\text{ or }\; P|_I = 1) \quad \text{and} \quad I \text{ contains only propagation literals} \\[1em]
&\mathsf{End0}: && (P,\, N,\, I,\, M) \leadsto_{\mathsf{End0}} M \quad \text{iff} \\
& && (0 \in P|_I \;\text{ or }\; N|_I = 1) \quad \text{and} \quad I \text{ contains only propagation literals}
\end{aligned}$$

Figure 9.2: Rules in Abstract Dual #DPLL. P and N are CNF formulae over V such that $N \equiv \neg P$, I and I' are interpretations over disjoint sets of variables, ℓ is a literal, and $M \in \mathbb{N}$. The procedure is based on DPLL combined with naive backtracking and terminates as soon as the entire search space has been processed.

terminal state is the model count of P and therefore of F. We denote with ℓ^d a *decision literal*, i. e., a literal which was assigned a value by a decision. Analogously, ℓ denotes a *propagation literal*. The rules are presented in Figure 9.2.

9.4.2 *Rules*

Dec I is extended by an unassigned decision literal ℓ^d whose variable occurs in V.

NB1 Naive backtracking is applied whenever the working interpretation either satisfies P or falsifies N and contains a decision literal ℓ^d. In this case, the working interpretation has the form $I\ell^d I'$. The model count is incremented by $2^{|V|-|I\ell^d I'|}$, according to rules R2 and R3 specified in Section 9.3.

NB0 Naive backtracking is applied whenever the working interpretation either satisfies N or falsifies P and contains a decision literal
decidedℓ. In this case, the working interpretation is of the form $I\ell^d I'$. The model count remains unaffected, see rules R1 and R4 specified in Section 9.3.

End1 The procedure terminates with a satisfying interpretation I. This is the case when I either satisfies P or falsifies N and contains no decision literal. The model count is incremented by $2^{|V|-|I|}$, according to rules R2 and R3 specified in Section 9.3.

End0 The procedure terminates with a falsifying interpretation I. This is the case when I either satisfies N or falsifies P and contains no decision literal. The model count remains unaffected, according to rules R1 and R4 specified in Section 9.3.

9.4.3 *Unit Propagation*

Unit propagation is simulated by the Dec and NB1 or NB0 rule, respectively, according to Section 9.3. In particular, the Dec rule may be applied if a unit clause $\{\ell\}$ occurs in either $P|_I$ or $N|_I$.

Unit Propagation in $P|_I$ Let's assume that after setting ℓ to 1, during the further execution of the procedure a conflict or empty reduct results in either $P|_I$ or $N|_I$. Naive backtracking is performed, M is updated, and ℓ's value is flipped according to rules NB1 or NB0 (see Section 9.4.2). Since ℓ is a unit literal in $P|_I$, the unit clause $\{\ell\}$ in $P|_I$ becomes empty when ℓ's value is flipped, and $I\overline{\ell}$ falsifies P. Hence, $\#P|_{I\overline{\ell}} = 0$. This corresponds to the second term in the sum of Equation 9.1.

Unit Propagation in $N|_I$ Let's assume that after setting ℓ to 1, during executing the procedure a conflict or empty reduct results in either $P|_I$ or $N|_I$. Naive backtracking is performed, M is updated, and ℓ's value is flipped according to one of the rules NB$\top$ or NB0 (see Section 9.4.2). Since ℓ is a unit literal in $N|_I$, the unit clause $\{\ell\}$ in $N|_I$ becomes empty, and $I\overline{\ell}$ falsifies N. Hence, $I\overline{\ell}$ satisfies P, and $\#P|_{I\overline{\ell}} = 2^{|V|-|I\overline{\ell}|}$. This corresponds to the second term in the sum of Equation 9.2.

9.4.4 *Soundness*

To make sure that the correct model count is returned by our framework, every node in the search tree must be visited or counted exactly once. By this we mean that an ancestor u of a node v may be visited, but not v itself. This can only occur if u represents a satisfying or falsifying interpretation. In this case, all its descendants, including v, represent a satisfying or falsifying interpretation as well, and the model count determined in v includes all of them. The naive backtracking mechanism employed in our DPLL version ensures that each node in the search tree is visited at most once. Therefore, we have to show that our framework does not allow for multiple visits of nodes.

The working interpretation I is extended iteratively by the Dec rule until it either satisfies or falsifies one of P or N. If I contains a decision literal, this indicates that not all combinations of truth values of the values have been tested yet. Chronological backtracking occurs and the value of the decision literal is flipped, i.e., another combination of truth values will be tested in the next step. The NB1 and NB0 rules describe this behaviour for the case in which I is either a satisfying or falsifying interpretation, respectively. Analogously, if I contains no decision literal, all possible combinations of truth values have been tested, and the search terminates. This behaviour is addressed by the End1 and End0 rules, where I is either a satisfying or falsifying interpretation, respectively.

To prove that our framework returns the correct model count, we show that the rules presented in Section 9.4.2 update the working model count correctly. The Dec rule extends the working interpretation I by a decision literal whenever the working interpretation neither satisfies nor falsifies either P or N, thus it must not alter the number of models. In our framework, M remains unchanged when the Dec rule is applied, and the requirement holds. The NB1 and End1 rules are applicable, whenever the working interpretation either falsifies N or is a (possibly partial) model for P. Whenever it falsifies N, it satisfies P and can be extended to 2^m models of P with $m = |V| - |I|$ denoting the number of unassigned variables. If prior to applying one of these two rules the working model count was M, it should amount to $M + 2^m$ afterwards. The NB1 and End1 rules are defined accordingly. The NB0 and End0 rules are applicable, whenever the working interpretation either falsifies P or is a model for N. In both cases, it can not satisfy P and thus can not be extended to a model of P. The working model count has to remain unaffected by the application of these two rules. In our framework, this is ensured by their definition.

From these arguments it follows that our framework is sound. We further implemented the framework in SWI-Prolog [199] making use of predicates defined in PIE [198] for a second check of the rules. First experiments showed the suitability of our approach, while a broader evaluation is ongoing.

9.5 EXAMPLE

We demonstrate the function of Abstract Dual #DPLL by an example. Let us consider a #SAT algorithm implementing the rules defined in Abstract Dual #DPLL introduced in Section 9.4 as well as rules UPP and UPN addressing unit propagation in P and N, respectively. Unit propagation in P can be executed if a unit clause occurs in $P|_I$, and, according to Equation 9.1, the model count is not affected. Unit propagation in N can be applied if a unit clause occurs in $N|_I$. It modifies the model count as shown in Equation 9.2. We define rules for unit propagation using the notation of our framework to illustrate their effect. To this end, we introduce transition relations $\rightsquigarrow_{\mathsf{UPP}}$ and $\rightsquigarrow_{\mathsf{UPN}}$, respectively.

$$\begin{array}{llll} \mathsf{UPP:} & (P,\, N,\, I,\, M) \rightsquigarrow_{\mathsf{UPP}} (P,\, N,\, I\ell,\, M) & \text{if} & \{\ell\} \in P|_I \\ \mathsf{UPN:} & (P,\, N,\, I,\, M) \rightsquigarrow_{\mathsf{UPN}} (P,\, N,\, I\ell,\, M + 2^{|V|-|I\ell|}) & \text{if} & \{\ell\} \in N|_I \end{array}$$

Let F be an arbitrary formula over a set of variables V, P and N be CNF formulae over V such that $P \equiv F$ and $N \equiv \neg F$, respectively. Let I denote an interpretation over V and M the working model count. The empty formula is represented by $\emptyset$, the empty clause by ε. In this context, I is represented by a sequence of literals.

Consider as an example $F = (x_1 \wedge x_2) \vee (x_3)$, where $\#F = 5$. Then, $V = \{x_1, x_2, x_3\}$, $P = (x_1 \vee x_3) \wedge (x_2 \vee x_3)$, and $N = (\neg x_1 \vee \neg x_2) \wedge (\neg x_3)$. We assume that the variables are ordered in the following manner: $x_1 < x_2 < x_3$. For choosing the decision literal, various heuristics may be applied. The same applies to the choice of the unit literal, if several unit clauses occur. In our example we define that literals are picked in ascending order of their variable and that they are set to $\top$. The execution trace is depicted in Table Table 9.1. Each row corresponds to an execution step with I, $P|_I$, $N|_I$, and M obtained by applying the rule indicated in the second column.

Step 0 The system is initialized: $P \equiv F$, $N \equiv \neg F$, $I = \varepsilon$, and $M = 0$. $N|_I$ contains a unit clause, namely $(\neg x_3)$, and I neither satisfies nor falsifies either P or N. Hence, the preconditions of the UPN rule are met.

Step 1 By means of the UPN rule, $\neg x_3$ is propagated and appended to I which becomes $I = \neg x_3$. M has to be increased by 2^m, where $m = |\{x_1, x_2, x_3\}| - |(\neg x_1)| =$

Table 9.1: Trace of a #SAT algorithm implementing the rules of Abstract Dual #DPLL extended by unit propagation. P and N are formulae such that $N \equiv \neg P$. I, $P|_I$, $N|_I$, and M denote the working interpretation, the working formulae and the working model count after applying the rule indicated in the second column, respectively. I is built according to the DPLL mechanism. Instead of terminating when I satisfies P, the model count is updated, naive backtracking is applied and the search continues until all decision literals are worked on.

STEP	RULE	I	$P\|_I$	$N\|_I$	M
0		ε	$(x_1 \vee x_3) \wedge (x_2 \vee x_3)$	$(\neg x_1 \vee \neg x_2) \wedge (\neg x_3)$	0
1	UPN	$\neg x_3$	$(x_1) \wedge (x_2)$	$(\neg x_1 \vee \neg x_2)$	4
2	UPP	$\neg x_3 \, x_1$	(x_2)	$(\neg x_2)$	4
3	UPP	$\neg x_3 \, x_1 \, x_2$	1	0	4
4	End$\top$	$\neg x_3 \, x_1 \, x_2$	1	0	5

2: $M = 4$. I neither satisfies nor falsifies either P or N, and $P|_I$ contains two unit clauses, namely (x_1) and (x_2), and the preconditions of the UPP rule are met.

Step 2 According to our heuristic, we choose x_1 and propagate it by means of the UPP rule. $I = \neg x_3 x_1$, M remains unaltered. I neither satisfies nor falsifies either P or N. Both $P|_I$ and $N|_I$ contain a unit clause each, namely (x_2) and $(\neg x_2)$, respectively, and the preconditions of UPP and UPN are met.

Step 3 We choose to propagate x_2 by means of the UPP rule. $I = \neg x_3 x_1 x_2$, M remains unaltered. I satisfies P and falsifies N. Since I contains no decision literals, the preconditions of both End1 and End0 are met.

Step 4 The search terminates with $I = \neg x_3 x_1 x_2$ satisfying P and falsifying N with the application of the End$\top$ rule. All variables are assigned a value, M is increased by $2^0 = 1$, and $M = 5$ is returned.

To assess the efficiency of an algorithm based on our framework, we use the number of rules applied as performance measure without counting the initialization step. The shorter an execution trace results, the better the algorithm performs. The execution trace of our example has length 4.

We now compare our dual approach with a non-dual one. To this end, let us consider a #SAT algorithm implementing the rules defined in Abstract #DPLL (see Section 9.2.4) as well as a rule UP addressing unit propagation. We introduce the transition relation $\rightsquigarrow_{\mathsf{UP}}$.

$$\boxed{\mathsf{UP}\colon\ (P,\, I,\, M) \rightsquigarrow_{\mathsf{UPP}} (P,\, I\ell,\, M) \quad \text{if} \quad \{\ell\} \in P|_I}$$

The execution trace of the non-dual algorithm executed on our previous example is depicted in Table 9.2.

Step 0 The system is initialized: $P \equiv F$, $I = \varepsilon$, and $M = 0$. I neither satisfies nor falsifies P, and the preconditions of the Dec rule are met.

Step 1 The Dec rule is applied. According to our heuristics, x_1 is chosen, set to 1 and appended to I which becomes $I = sequence{x_1}^d$. The model count remains

Table 9.2: Trace of a #SAT algorithm implementing the rules of Abstract #DPLL extended by unit propagation. I, $P|_I$ and M are the working interpretation, formula and model count, respectively. During execution, I is built according to the DPLL mechanism. Instead of terminating when I satisfies P, the model count is updated, naive backtracking is applied and the search continues until all decision literals are worked on.

STEP	RULE	I	$P\|_I$	M
0		ε	$(x_1 \vee x_3) \wedge (x_2 \vee x_3)$	0
1	Dec	${x_1}^d$	$(x_2 \vee x_3)$	0
2	Dec	${x_1}^d\,{x_2}^d$	1	0
3	NB⊤	${x_1}^d \neg x_2$	(x_3)	2
4	UP	${x_1}^d \neg x_2\, x_3$	$\emptyset$	2
5	NB⊤	$\neg x_1$	$(x_3) \wedge (x_2 \vee x_3)$	3
6	UP	$\neg x_1\, x_3$	$\emptyset$	3
7	End⊤	$\neg x_1\, x_3$	$\emptyset$	5

unaltered. I neither satisfies nor falsifies P, and the preconditions of the Dec rule are met.

Step 2 The Dec rule is applied. According to our heuristics, x_2 is chosen, $I = {x_1}^d \neg x_2$, and M remains unaltered. Since I satisfies P and contains two decision literals, namely ${x_1}^d$ and ${x_2}^d$, the preconditions of the NB⊤ rule are met.

Step 3 Naive backtracking is applied. The number of unassigned variables amounts to 1, and the model count is increased by $2^1 = 2$ resulting in $M = 2$. The value of the decision literal ${x_2}^d$ is flipped, i.e., $x_2 = 0$, and x_2 is turned into a propagation literal. $I = {x_1}^d \neg x_2$ neither satisfies nor falsifies P. Since $P|_I$ contains the unit clause (x_3), the preconditions of the UP rule are met.

Step 4 Unit propagation is applied by setting x_3 to 1. $I = {x_1}^d \neg x_2\, x_3$, and $P|_I = 1$. I satisfies P and still contains a decision literal, namely ${x_1}^d$, hence the preconditions of the NB1 rule are met.

Step 5 Naive backtracking is applied. All variables were assigned a value, and the model count becomes $M = 3$. $I = \neg x_1$ neither satisfies nor falsifies P, and P contains the unit clause (x_3). The preconditions of the UP rule are met.

Step 6 The UP rule is applied by propagating x_3. The working interpretation becomes $I = \neg x_1\, x_3$. It satisfies P and contains no decision literal meeting the preconditions of the End1 rule.

Step 7 The execution terminates with $I = \neg x_1\, x_3$ satisfying P. The number of unassigned variables is 2, and $M = 3 + 2 = 5$ is returned.

The execution trace has length 7. We show that its length depends on the decision heuristics applied. Let the order in which the decision literals are chosen be reversed. Then, in Step 1, x_3 is chosen as decision literal. After backtracking in Step 2, x_3 is set to $\perp$ and $P|_I = (x_1) \wedge (x_2)$. In Step 3, x_2 is propagated by means of the UP rule, and $P|_I = (x_1)$. In Step 4, UP is applied and x_2 propagated, resulting in $P|_I = \emptyset$. The execution terminates in Step 5 with the application of the End1 rule, and $M = 5$ is returned. The execution trace has length 5.

9.6 CONCLUSION AND FUTURE WORK

The problem #SAT of determining the number of models of a propositional formula has many real-world applications. In this work, we have presented a formal framework describing a #SAT solving procedure based on DPLL, called Abstract Dual #DPLL, including a formalization of a non-dual variant, called Abstract #DPLL, and argued that our framework is sound. The Abstract Dual #DPLL procedure is given by 5 simple rules which specify the decide and naive backtracking mechanisms. The application of chronological backtracking underlying naive backtracking and the framework's compactness facilitate the investigation of the main idea, namely to consider a formula and its negation simultaneously in #SAT solving. We demonstrated the working of Abstract Dual #DPLL on an example assuming an implementation enhanced by unit propagation and compared it do a non-dual algorithm based on Abstract #DPLL enhanced by unit propagation as well. The dual algorithm performed better,i. e., less rules were executed. This is due to the fact that in Step 1 unit propagation can be executed on $N|_I$, whereas in the non-dual version, a decision has to be taken. For every decision, at a later time point backtracking occurs. This results in a longer execution trace. In this example, the performance of the dual version does not depend from the decision heuristics applied, contrarily to the non-dual version.

Today, several #SAT solvers are available. They implement various strategies, however, to our best knowledge, no dual approach has been presented yet. We implemented our framework in SWI-Prolog, and first experiments on small crafted formulae are encouraging. An interesting question is whether by Abstract Dual #DPLL state-of-the-art #SAT solvers can be modeled. relsat v2.00 [15] is based on DPLL, but contrarily to our framework splits the formula under consideration into subformulae over disjoint variable sets. At present, we can not model #SAT solving procedures making use of backjumping or CDCL, such as Cachet [172], since non-chronological backtracking and clause learning are not supported. The performance gain of some modern #SAT solvers is due to improved data structures. This aspect is not covered by our framework as we focus on algorithms. In order to model countAtom [37], our framework should be extended to support parallelization.

For the sake of simplicity we assume that we are given two formulae P and N over the same set of variables which are duals of each others, e.g., models of P falsify N and vice versa. This assumption is rather strong unless we allow additional variables, e.g., Tseitin variables, to encode negation [192]. Let F be an arbitrary formula over variables V. We denote with $T(F)$ the Tseitin transformation of F. The models of $T(F)$ projected onto V are exactly the models of F. Therefore, our approach can be generalized to the situation in which N contains, e.g., Tseitin variables, by projecting the models of N onto the variables occurring in P.

As future work, we will make our soundness arguments more precise and investigate completeness. We also plan a more extensive experimental evaluation and a detailed comparison of our dual approach with non-dual methods. We intend to extend our work to the case where the formula under consideration and its negation communicate over "inputs" by allowing, e.g., Tseitin variables. Finally, by extending our framework to model strategies implemented in state-of-the-art SAT solvers, such as conflict-driven clause learning (CDCL) [128], we want to combine the strengths of duality of our Abstract Dual #DPLL with the strength of modern SAT solvers to obtain a state-of-the-art model counting framework.

10

DISCUSSION OF PAPER 1

In Section 10.1, the main contributions are emphasized. We assume the input formula and its negation be defined over the same set of variables. Consequently, their residuals under a given trail are correlated (Section 10.2). Our framework allows for computing the models and the model count of the input formula considering exclusively its negation. We provide an example (Section 10.3), and finally in Section 10.4 we show how the models of a propositional formula can be computed by means of Abstract Dual #DPLL and provide an example.

10.1 MAIN CONTRIBUTIONS

This paper presents two major contributions with potential to speed up model counting. First, as we saw in Section 9.5, unit propagation in the negation of the input formula saves decisions. In our example this fact did not come out clearly, and we want to catch up on this by a simple example.

Example 10.1 (Dual reasoning saves decisions). *Consider the propositional formula $F = (x_1 \vee \ldots \vee x_n)$ defined over the set of variables $V = \{x_1, \ldots, x_n\}$.*[1] *It is already in CNF, therefore we set $P = F$. We assume the variables to be ordered in ascending order with respect to their index, i. e., $x_1 < \ldots < x_n$, and to assign decision literals the value 1. In non-dual mode, we take the decision ${x_1}^d$, which is a partial model of F representing 2^{n-1} total models of F. We backtrack chronologically and flip ${x_1}^d$, i. e., set x_1 to 0. The residual of P under $\neg x_1$ is $P|_{\neg x_1} = (x_2 \vee \ldots \vee x_n)$. It is undefined, and we take the decision ${x_2}^d$, which again satisfies F and represents 2^{n-2} total models of F, followed by flipping ${x_2}^d$ and repeating these steps until finally x_n can be propagated. The negation of F is $\neg F = (\neg x_1) \wedge \ldots \wedge (\neg x_n)$. It is already in CNF, therefore we set $N = \neg F$. Obviously, the formula N can be solved using only unit propagation. In dual mode, we first propagate $\neg x_1$ by means of rule UPN and find 2^{n-1} models. We then propagate $\neg x_2$ finding 2^{n-2} models and continue until all unit literals have been propagated.*

Second, dual reasoning enables finding partial models on the fly without the need for exploring and shrinking total models. Modern SAT solvers on the one hand usually only detect total models. On the other hand, they are able to detect partial assignments falsifying the formula in question. By considering the negation of the input formula, we exploit this fact: Assume a partial assignment satisfies the input formula. Necessarily unit propagation has been carried out until completion. A state-of-the-art SAT solver extends this partial assignment to a total satisfying assignment without encountering a conflict. This is computationally

1 The notation coincides with the one introduced in Part II: variables and literals are denoted by lowercase letters, possibly with subscripts, the empty clause is denoted by 0 and the empty CNF formula by 1, 0 denotes $\bot$, 1 denotes $\top$, and x^d denotes the decision $\dot{x}$.

Table 10.1: Trace of Abstract Dual #DPLL with unit propagation finding partial models.

STEP	RULE	I	$P\vert_I$	$N\vert_I$	M
0		ε	P	N	0
1	Dec	a^d	$(\neg c) \wedge (\neg d) \wedge (b \vee \neg c) \wedge (b \vee \neg d)$	$(c \vee d)$	0
2	UPP	$a^d \neg c$	$(\neg d) \wedge (b \vee \neg d)$	(d)	0
3	UPP	$a^d \neg c \neg d$	1	0	0
4	NB⊤	$\neg a$	$(b \vee \neg c) \wedge (b \vee \neg d)$	$(\neg b) \wedge (c \vee d)$	2
5	UPN	$\neg a \neg b$	$(\neg c) \wedge (\neg d)$	$(c \vee d)$	6
6	UPP	$\neg a \neg b \neg c$	$(\neg d)$	(d)	6
7	UPN	$\neg a \neg b \neg c\, d$	0	1	7
8	End⊥	$\neg a \neg b \neg c\, d$	0	1	7

less expensive than checking whether the partial assignment satisfies the input formula, and it makes sense in SAT solving.[2] In #SAT, however, the search space need be processed exhaustively, and more expensive techniques might pay off.

10.2 CORRELATION OF THE RESIDUALS

In our paper we assume that P and N are computed from F and $\neg F$ without introducing auxiliary variables. This is a rather strong assumption and leads to two observations: First, P and N might be exponentially larger than F and $\neg F$.[3] Second, we distinguish three cases, given a (possibly partial) trail I: 1) $P|_I = 1$ and $0 \in N|_I$, 2) $0 \in P|_I$ and $N|_I = 1$, 3) both $P|_I$ and $N|_I$ are undefined. This behavior was already observed in the example in Section 9.5 and holds in general for Abstract Dual #DPLL. This is no coincidence but an obvious consequence of the fact that $\mathsf{var}(P) = \mathsf{var}(N)$, and that N and P are the negation of each other. However, Abstract Dual #DPLL enables the detection of partial models demonstrated by the following example, which also shows the correlation of $P|_I$ and $N|_I$.

Example 10.2 (Correlation of $P|_I$ and $N|_I$ in Abstract Dual #)**.** *Let $V = \{a, b, c, d\}$ be a set of propositional variables and $F = (\neg a \wedge b) \vee (\neg c \wedge \neg d)$ be an arbitrary formula defined over V. Its CNF transformation is obtained by applying the distributivity laws and is given by $P = P_1 \wedge P_2 \wedge P_3 \wedge P_4 = (\neg a \vee \neg c) \wedge (\neg a \vee \neg d) \wedge (b \vee \neg c) \wedge (b \vee \neg d)$. The negation of F is already in CNF, and $N = N_1 \wedge N_2 = (a \vee \neg b) \wedge (c \vee d)$. The total models of F are $a\, b \neg c \neg d$, $a \neg b \neg c \neg d$, $\neg a\, b\, c\, d$, $\neg a\, b\, c \neg d$, $\neg a\, b \neg c\, d$, $\neg a\, b \neg c \neg d$, and $\neg a \neg b \neg c \neg d$, and $\#F = 7$. The execution steps are listed in Table 10.1. After deciding a (step 1) and propagating $\neg c$ with reason P_1 (step 2) and $\neg d$ with reason P_2 (step 3), the residual of P under the resulting trail $a^d \neg c \neg d$ becomes empty and a conflict in N occurs. The detected model is partial and represents two total models. The model count is updated, the model counter backtracks and flips the most recent decision a^d (step 4). After backtracking, we have $I = \neg a$, and both $P|_I$ and $N|_I$ are undefined, but $N|_I$ contains the unit clause $(\neg b)$. Propagating $\neg b$ with reason N_1 results in the trail $\neg a \neg b$ in which two variables are unassigned. It represents four total assignments, and their count is summed up to the*

2 The SMT solver SPASS-SATT [33, 34] implements clause watching. It executes satisfiability checks before taking a decision and is therefore able to detect partial models.

3 This issue is solved in Chapter 11.

Table 10.2: Execution steps for Abstract Dual #DPLL considering only the negated formula.

STEP	RULE	I	$N\|_I$	M	DETECTED MODEL
0		ε	$(a \vee \neg b) \wedge (c \vee d)$	0	
1	Dec	a^d	$(c \vee d)$	0	
2	Dec	$a^d\, c^d$	1	0	
3	NB⊥	$a^d \neg c$	(d)	0	
4	UPN	$a^d \neg c\, d$	1	2	$m_1 = a \neg c \neg d$
5	NB⊥	$\neg a$	$(\neg b) \wedge (c \vee d)$	2	
6	UPN	$\neg a \neg b$	$(c \vee d)$	6	$m_2 = \neg a\, b$
7	Dec	$\neg a \neg b\, c^d$	1	6	
8	NB⊥	$\neg a \neg b \neg c$	(d)	6	
9	UPN	$\neg a \neg b \neg c\, d$	1	7	$m_3 = \neg a \neg b \neg c \neg d$
10	End⊥	$\neg a \neg b \neg c\, d$	1	7	

model count M (step 5). After propagating $\neg c$ with reason P_3 (step 6) and d with reason N_2 (step 7), a conflict in N occurs and the resulting trail satisfies P. The trail contains no decisions and represents a total model of F, and the search terminates with $M = 7$.

10.3 MODEL COUNTING USING ONLY THE NEGATED FORMULA

The strength of our dual approach lies in the fact that in some cases we may choose whether to execute a rule in P or in N. Obviously, one would apply unit propagation as long as possible, in whichever formula the unit occurs. While it is obviously not optimal, as we will show in Example 10.3, it is interesting to notice that the models of F can also be counted by considering only N and the rules Dec and UPN as well as NB⊤, NB⊥, End⊤, and End⊥. The formulation of variants of the termination and backtracking rules neglecting P and the according state representation are straightforward, and to keep the presentation short, we refrain from providing them. Also, our rules do not prioritize rule execution in P over rule execution in N. Therefore, the rule set need not be adapted, and neither does the representation of the state. It suffices to ignore the conditions on P.

Example 10.3 (Consider only the negated formula in Abstract Dual #DPLL.). *Consider again Example 10.2, where $F = (\neg a \wedge b) \vee (\neg c \wedge \neg d)$, which is defined over the set of variables $V = \{a, b, c, d\}$, and $N = N_1 \wedge N_2 = (a \vee \neg b) \wedge (c \vee d)$. We only consider the rules Dec and UPN as well as NB⊤, NB⊥, End⊤, and End⊥ restricted as described above. The execution steps are listed in Table 10.2. Ignore the last column for now, we will come back to it in Section 10.4 and Example 10.4.*

After deciding a (step 1) and c (step 2), a model of F is found. Backtracking occurs and the decision c^d is flipped (step 3) upon which the literal d is propagated with reason N_2 (step 4). The trail $a^d \neg c\, d$ represents two total assignments, and M is incremented by 2. The trail also satisfies N, i.e., it is a counter-model of F. Backtracking occurs (step 5), followed by propagating $\neg b$ with reason N_1. The resulting trail $\neg a \neg b$ represents four total assignments, and M is incremented by four (step 6). The residual of N under the current trail contains no unit, a decision is taken (step 7), and N is satisfied. After backtrack-

ing (step 8) and propagation of d with reason N_2, the trail $\neg a \neg b \neg c d$ represents one total assignment, and M is incremented by one. The trail satisfies N, and since it contains no decision, the computation terminates with $M = 7$.

The execution in Table 10.2 considering only N requires more steps than the execution taking into account both P and N in Table 10.1. This behavior was to be expected, since—just like in non-dual counting—the model counter can not exploit the presence of units in $P|_I$. It has to take a decision and to flip it later.

While we are interested in finding conflicts in N, unit propagation in N tries to extend the current trail to a model of N. If we would decide the negation of this unit, we would immediately obtain a conflict. Notice that we can not propagate the negation of a unit in N, because N contains no reason for it. By propagating the unit and counting the number of models we would find if we would decide its opposite value and backtrack, this work is saved.

10.4 COMPUTING MODELS

In Abstract Dual #DPLL, we count the models of a propositional formula. However, we can also compute them: If the current trail I satisfies P, the model is I. Similarly, if I falsifies N, it is a counter-model of $\neg F$ and hence a model of F. The interesting case is unit propagation in N by means of the UPN rule: when propagating a literal ℓ in $N|_I$, we count the models we would have found if we had decided $\neg\ell$, obtained a conflict in $N|_{I\neg\ell^d}$, and backtracked chronologically. Therefore, these models are all total extensions of $I\neg\ell$. The discussion in Section 9.3 and the definition given in Equation 9.2 led to rule UPN.

Example 10.4 (Computing models in Dual Abstract #DPLL)**.** *Consider again Example 10.3, where $F = (\neg a \wedge b) \vee (\neg c \wedge \neg d)$ and $V = \{a, b, c, d\}$. The last column of Table 10.2 contains the (implicitly) detected models. Notice that we do not enumerate them, but the rules of Abstract Dual #DPLL can be adapted accordingly. It can easily be verified that $m_1 = a \neg c \neg d$ counted in step 4 is a model of F. It is a partial model and represents two total models, namely $a b \neg c \neg d$ and $a \neg b \neg c \neg d$. Also $m_1 = \neg a \neg b$ counted in step 6 satisfies F. It represents four total models, namely $\neg a b c d$, $\neg a b c \neg d$, $\neg a b \neg c d$, and $\neg a b \neg c \neg d$. Finally, $m_3 = \neg a \neg b \neg c \neg d$ is a total model of F. A comparison with Example 10.2 confirms that all models of F have been found.*

11

PAPER 2: DUALIZING PROJECTED MODEL COUNTING

PUBLISHED. In: *IEEE 30th International Conference on Tools with Artificial Intelligence, ICTAI 2018, 5–7 November 2018, Volos, Greece*. Ed. by Lefteri H. Tsoukalas, Éric Grégoire, and Miltiadis Alamaniotis. IEEE, 2018, pp. 702–709. DOI: 10.1109/ICTAI.2018.0011.

AUTHORS. Sibylle Möhle and Armin Biere.

ACKNOWLEDGMENTS. Supported by the Austrian Science Fund (FWF) grant S11408-N23 (RiSE).

ABSTRACT. In many recent applications of model counting not all variables are relevant for a specific problem. For instance redundant variables are added during formula transformation. In projected model counting these redundant variables are ignored by projecting models onto relevant variables. Inspired by dual propagation which has its origin in solving quantified Boolean formulae and jointly works on both the original formula and its negation, we present a novel calculus for dual projected model counting. It allows to capture existing techniques such as blocking clauses, chronological as well as non-chronological backtracking, but also introduces new concepts including discounting and dual conflict analysis to obtain partial models. Experiments demonstrate the benefit of our approach.

11.1 INTRODUCTION

Classical applications of #SAT, the task of counting the models of a propositional theory, are found in the area of probabilistic reasoning [167] adopted in, e.g., medical diagnosis and planning. Further application scopes encompass circuit design [36] and quantitative information flow analysis [26] as well as differential cryptanalysis [106]. In product configuration, the number of valid configurations under given preconditions might be of interest [110]. For some tasks not all configuration options might be relevant, thus only the models projected onto the relevant variables are counted [203]. Projected models may be required in planning as well [10, 202]. The task of *projected model counting* is also referred to as #∃SAT, since the main idea of projection is to existentially quantify the irrelevant variables, i.e., the variables which are to be discarded. In this sense, #SAT is a special case of #∃SAT, namely the one in which the set of irrelevant variables is empty [202].

Particular challenges in model counting. In propositional model counting, contrarily to SAT solving, the search does not terminate after the detection of a model. Instead, the search space needs to be explored exhaustively. Conflict Driven Clause

Learning (CDCL) [128, 146] provides efficient means to deal with conflicting assignments, but a comparable method for handling satisfying assignments is still not available. Some state-of-the-art #SAT solvers prune the search space upon finding a satisfying assignment by adding blocking clauses with the aim to prevent multiple model counts [104]. An apparent drawback of this approach is a substantial growth of the formula since these blocking clauses are irredundant and therefore must not be deleted. This issue is addressed in [94], a solver for projected model enumeration working without blocking clauses, and in [84] where blocking clauses are eagerly deleted and the number of kept blocking clauses is at any time limited to be at most linear in the number of relevant variables. Although devised for projected answer set enumeration, this method is readily applicable for #∃SAT. The addition of large clauses may furthermore slow down the solver. On this account, in #CLASP [10] the blocking clauses are minimized which on the one hand reduces their count and on the other hand prunes a larger portion of the search space. To detect partial models, the DPLL-based model counter CDP [27] performs satisfiability checks on the residual of the formula. This is not the case in state-of-the-art SAT solvers which only keep track of the assigned variables. From a complexity point of view this makes sense in SAT, since a satisfying assignment can always be extended to a total model by a number of decisions linear in the variable count and thus compensates for the overhead introduced by, e.g., clause watching. In contrast, detecting partial models and subsequent pruning of the search space leads to a higher performance gain in #SAT, hence more computational effort may be invested. For the same reason, more involved preprocessing techniques are justified in model counting [115].

Our motivation. Inspired by [72], Abstract Dual #DPLL [24] was developed with the aim to investigate the suitability of a dual approach for exact model counting as an alternative to component analysis [15, 173, 189] which can be considered the state of the art in exact #SAT solving. One objective of our work presented here is to extend this approach to facilitate projected model counting. A second goal is to incorporate methods which are widely used in state-of-the-art SAT solvers. Our third goal is to investigate the impact of a dual approach on the performance of a non-dual method. We only partially consider preprocessing [65, 115] in this work, but plan to further investigate its applicability in a dual setting and to develop adequate methods in the future.

Our contribution. In this work we present the first dual calculus addressing projected exact unweighted model counting. First, we present a dual representation of the formula under consideration which facilitates the detection of partial models and subsequent pruning of the search space. Our method incorporates "good learning" and is exempt from satisfiability checks and clause watching mechanisms. This results in a significant performance gain compared to the non-dual variant of the same algorithm which finds only total models. Second, our calculus takes an arbitrary formula or a circuit as argument and returns the correct model count even if for its transformation into CNF techniques are adopted which in general do not preserve the model count, such as the Plaisted-Greenbaum transformation [164]. Third, we introduce a novel technique based on *flipping* and *discounting* which prevents multiple model counts without the use of blocking clauses. Our calculus models techniques implemented in state-of-the-art SAT solvers, such as conflict analysis and conflict-driven backjumping. Finally, we provide a robust and carefully tested implementation DUALIZA of our calculus.

```
$ cat clause.form
a | b | c | d
$ dualiza -e clause.form
ALL SATISFYING ASSIGNMENTS
d
c !d
b !c !d
a !b !c !d
$ dualiza clause.form
NUMBER SATISFYING ASSIGNMENTS
15
```

```
$ dualiza clause.form -l | grep RULE
c LOG 1 RULE UNX 1      -4
c LOG 1 RULE UNX 2      -4
c LOG 1 RULE BN0F 1     -4
c LOG 2 RULE UNX 3      -3
c LOG 2 RULE BN0F 2     -3
c LOG 3 RULE UNX 4      -2
c LOG 3 RULE BN0F 3     -2
c LOG 3 RULE UP 1       1
c LOG 3 RULE EP1 1
```

Figure 11.1: On the left hand side of the figure our implementation DUALIZA is applied to a simple example consisting of a single clause. The "log" output on the right hand side demonstrates that in dual mode this example is solved in essence by just dual propagation (UNX) (see Section 11.6 for details).

In Figure 11.1 we provide a simple example to highlight the power of dual projected model counting. More details are discussed in Section 11.6.

The paper is organized as follows: In Section 11.2, we discuss basic concepts. Section 11.3 introduces the concept of duality and our new duality property. Our calculus is presented in Section 11.4 on which our implementation described in Section 11.5 is based. After experiments in Section 11.6, we discuss related work in Section 11.7 before we conclude in Section 11.8.

11.2 PRELIMINARIES

Let F be an arbitrary (propositional) formula over variables Z, interpreted over Boolean constants $\mathbb{B} = \{0, 1\}$. Further assume Z to be partitioned into the set of *relevant input variables* X and the set of *irrelevant input variables* Y. We denote with *inputs* the variables contained in either X or Y. A total assignment $\sigma\colon Z \to \mathbb{B}$ maps Z to the truth values 1 and 0 and can be applied to F to yield a truth value $\sigma(F) \in \mathbb{B}$, also written $F|_\sigma$. A *relevant input assignment* is defined on X and undefined elsewhere. Similarly, an *irrelevant input assignment* is defined. This lets us decompose any total $\sigma = \sigma_X \cup \sigma_Y$ into its relevant σ_X and irrelevant σ_Y part. We use $F(U)$, or equivalently $\mathsf{var}(F) \subseteq Z$, to denote that F only depends on (contains) a subset $U \subseteq Z$ of all variables and write $F(X, Y)$ if F is defined over $X \cup Y$.

We count the number of satisfying assignments of $F(X, Y)$ projected onto the relevant variables X, defined as

$$\#\exists Y \,.\, F(X,Y) = \ |\{\tau\colon X \to \mathbb{B} \mid \text{exists } \sigma : Z \to \mathbb{B} \text{ with} \\ \sigma\big(F(X,Y)\big) = 1 \text{ and } \tau = \sigma_X\}|.$$

Thus #SAT turns out to be the special case where $Y = \emptyset$.

In order to make use of sophisticated data structures and algorithms used in modern SAT solvers we further consider propositional formulae in CNF and thus for instance apply Tseitin transformation [192] on F to obtain a CNF representation $P(X, Y, S)$ of F depending on X and Y as well as on variables S introduced by the transformation. Note that, even though the Tseitin encoding is only satisfiability-preserving, i.e., the result is not logically equivalent, it does preserve the number of models, assuming all inputs are relevant. The models of F projected onto X are exactly the models of its Tseitin representation $\exists S \,.\, P(X, Y, S)$ projected onto X, and

$$\#\exists\, Y, S \,.\, P(X, Y, S) = \#\exists\, Y \,.\, F(X, Y).$$

Our approach uses two formulae to capture the projected model counting problem. The *primal* formula $P(X, Y, S)$ ranges over the inputs X and Y and the *primal* variables S, while the *dual* formula $N(X, Y, T)$ ranges over the same inputs X and Y but also over the *dual* variables T instead of the primal variables S. The idea is that N is the "negation" of F, and hence of P, which is easy to achieve by encoding the negation of F using Tseitin variables. The precise condition is discussed further down. In line with the definition of an assignment, a *primal assignment* σ_S and a *dual assignment* σ_T are defined. This extends our notion of total assignment to $\sigma = \sigma_X \cup \sigma_Y \cup \sigma_S \cup \sigma_T$ over variables $V = X \cup Y \cup S \cup T$.

A CNF F over V is a conjunction of clauses and each clause is a disjunction of literals. A literal ℓ is either a variable $v \in V$ or its negation $\neg v$. In both cases we write $\mathsf{var}(\ell) = v$ and extend this notion to sequences and sets of literals as well as formulae. We also use $\overline{\ell} = \neg\ell$ and assume $\neg\neg\ell = \ell$. We also write $C \in F$ if C is a clause of F and similarly $\ell \in C$. A sequence $I = \ell_1 \cdots \ell_n$ of literals with mutually exclusive variables ($\mathsf{var}(\ell_i) \neq \mathsf{var}(\ell_j)$ for $i \neq j$) is called a *trail*. Trails can be concatenated, $I = I'I''$, assuming the variables in I' and I'' are distinct. We also use $\ell \in I$, treating I as set of literals, and in particular define $X - I = X - \mathsf{var}(I) \subseteq X$, the subset of variables of X not in I. These trails are also interpreted as partial assignments with $I(\ell) = 1$ iff $\ell \in I$. Thus $I(\ell) = 0$ if $\overline{\ell} \in I$ and $I(\ell)$ is undefined if $\mathsf{var}(\ell) \notin \mathsf{var}(I)$. This gives the useful shortcut $2^{|X-I|}$ to denote the number of (total) relevant input assignments covered by the partial assignment represented by the trail I. We denote the projection of I onto X by $\pi(I, X)$ and consider it also a conjunction of literals. The *residual* of F with respect to I, denoted by $F|_I$, is obtained by replacing in F the literals occurring in I by their truth value. I *satisfies* F, denoted by $I \models F$, if $F|_I = \emptyset$. Trail I *falsifies* F if it contains the empty clause, i.e., $\emptyset \in F|_I$.

We are going to present a proof calculus in the style of DPLL(T) [155] and will also include elements of CDCL formalized in [97]. Proof rules model state transitions of an abstract projected model counting DPLL / CDCL solver.

Besides primal and dual formulae, the main ingredient of an abstract state is a trail which, as discussed above, captures the current partial assignment and in addition marks some of its literals ℓ as decision literals, using the notation ℓ^d. These denote open "left" (or "first") branches in the sense of DPLL.

We also mark literals ℓ starting a "right" (or "second") branch with an f followed by the number of models found at the corresponding decision level in parenthesis, i.e., $\ell^{f(m)}$. The idea is that after closing the left branch of a decision its decision literal ℓ^d becomes a *flipped literal* $\overline{\ell}^{f(m)}$ starting the right branch of that decision maintaining its decision level $\mathsf{dl}(\ell)$. We denote the set of decision literals in I by $\mathsf{decs}(I)$ and interpret it as a conjunction of literals where appropriate. In an analogous manner, we represent the set of flipped literals in I by $\mathsf{flips}(I)$ and the set of unit literals in a residual $G|_I$ of a formula G w.r.t. I by $\mathsf{units}(G|_I)$.

Backjumping involves the addition of redundant clauses which may be deleted anytime. We mark redundant clauses with an r, writing C^r, to distinguish them from blocking clauses which prevent multiple model counts and therefore have to be retained until the end of the procedure.

11.3 DUALITY

In propositional model counting the search space needs to be processed exhaustively. In this sense #SAT exhibits a certain analogy to QBF solving where, due to the existence of universal quantifiers in the formula, the complete search space needs to be traversed.

To overcome this limitation, dual propagation was introduced for circuits in [91] and adapted to CNF-based QBF solving in [93]. This work inspired our abstract framework developed with focus on DPLL [24]. In the work reported here, we extend this approach in two ways: first by projection and second by elements of CDCL.

Let $F(X, Y)$ be an arbitrary (propositional) formula and our task is to compute the number of models of F projected onto X. Following the concepts introduced in Section 11.2, we compute a *dual representation* of F consisting of two CNF formulae $P(X, Y, S)$ and $N(X, Y, T)$. Recall that P and N encode F and its negation, respectively.

Our model counter executes a dual variant of CDCL on P and N simultaneously and maintains a single trail I with $\mathsf{var}(I) = X \cup Y \cup S \cup T$. The input variables in X and Y are shared by P and N and are called *shared variables*. They may be propagated in either formula [91].

Basic idea. Every trail I either satisfying P or falsifying N is a (partial) model of F and represents $2^{|X-I|}$ total models of F projected onto X. In contrast, no model of F can be computed if I falsifies P. Obviously, the same holds if I satisfies N but due to the structure of I this situation will never arise in our framework as will be clarified further down. Conflict analysis is performed after a conflict is detected, and backtracking occurs upon either a conflict or the detection of a model of F. If I is a partial assignment, backtracking prunes a potentially large portion of the search space. The procedure terminates as soon as the complete search space has been processed.

Thus in our approach we assume $F(X, Y)$ is represented by a pair of dual formulae satisfying the following definition.

Definition 11.1 (Combined Formula Pair). *A* combined formula pair *of a formula $F(X, Y)$ consists of formulae $P(X, Y, S)$ and $N(X, Y, T)$ meeting the following conditions:*

$$\exists\, S \,.\, P(X, Y, S) \equiv F(X, Y) \tag{11.1}$$
$$\exists\, T \,.\, N(X, Y, T) \equiv \neg F(X, Y) \tag{11.2}$$

where X, Y, S and T are pairwise disjoint sets of variables. We denote a combined formula pair of $F(X, Y)$ by $[P(X, Y, S) \parallel N(X, Y, T)](F)$ or $[P \parallel N](F)$.

Definition 11.1 essentially states that F and P are semantically equivalent, i.e., have the same models, upon projection onto $X \cup Y$ and that the same holds for $\neg F$ and N.

As a consequence, if P and N are a combined formula pair of F, for every total assignment σ of X and Y it holds that $\sigma(\exists S \,.\, P(X, Y, S)) \neq \sigma(\exists T \,.\, N(X, Y, T))$.

Definition 11.2 (Duality Property). *Let X, Y, S and T be pairwise disjoint sets of variables. Two formulae $G(X, Y, S)$ and $H(X, Y, T)$ comply with the* duality property*, if*

$$\forall X, Y \,.\, ((\exists\, S \,.\, G(X, Y, S)) \oplus (\exists\, T \,.\, H(X, Y, T))) \tag{11.3}$$

where "$\oplus$" denotes "exclusive or (XOR)".

Lemma 11.1. *The duality property holds for a combined formula pair* $[P(X,Y,S) \parallel N(X,Y,T)](F)$.

Consider an assignment I (trail) with $P(X,Y,S)|_{I'} \models I$ for $I' = \pi(I, X \cup Y)$, which for instance holds if variables in X and Y are the only decisions in I and the rest of I is obtained through unit propagation in P. Then the dual property in Equation 11.3 continues to hold for residuals w.r.t. I:

$$\forall X, Y \,.\, ((\exists\, S \,.\, P(X,Y,S)|_I) \oplus (\exists\, T \,.\, N(X,Y,T)|_I)) \tag{11.4}$$

This property provides the most important invariant for our calculus discussed in the next section, but requires that we split on X and Y variables first.

In general, we assume that only Equation 11.3 holds for P and N, without necessarily requiring that there exists an F satisfying Definition 11.1. In this situation, even after assigning X and Y and performing unit propagation, there might still be an unassigned variable $s \in S$ left. Assume we extend the trail with the decision ℓ with $\mathsf{var}(\ell) = s$. At this point the dual property (Equation 11.4) might seize to hold, since the value picked for s may lead to an unsatisfiable residual $P(X,Y,S)|_{I\ell}$ even if $P(X,Y,S)|_{I\bar{\ell}}$ is satisfiable, thus both $\exists\, S \,.\, P(X,Y,S)|_{I\ell}$ and $\exists\, T \,.\, N(X,Y,T)|_{I\ell}$ are false.

For correctness it is sufficient to first split on relevant variables X, followed by irrelevant variables Y and primal variables S, but never split on dual variables T. This splitting order maintains the following direction (Equation 11.5) of the dual property (Equation 11.4) on residuals, namely that a conflict in N guarantees that all extensions to the current assignment to relevant variables in X can be extended to total models of P:

$$\forall X, Y \,.\, ((\neg\exists\, T \,.\, N(X,Y,T)|_I) \rightarrow (\exists\, S \,.\, P(X,Y,S)|_I)) \tag{11.5}$$

A formal proof of this invariant and accordingly the correctness of our calculus is out of scope of this paper. For now we rely on extensive testing and thus an empirical justification.

11.4 CALCULUS

We describe the framework of our calculus by a labeled state transition system $\langle S, L, \rightsquigarrow, s_0 \rangle$ with set of states S, set of labels L and transition relation $\rightsquigarrow \subseteq S \times S$. Intermediate states are of the form $(P,\, N,\, I,\, M)$ where $P(X,Y,S)$ and $N(X,Y,T)$ are a combined formula pair of $F(X,Y)$, I is a trail with $\mathsf{var}(I) \in X \cup Y \cup S \cup T$ and $M \in \mathbb{N}$. The initial state is defined by $s_0 = (P,\, N,\, (),\, 0)$ with $()$ denoting the empty trail. The terminal state is $M \in \mathbb{N}$ representing $\#\exists Y \,.\, F(X,Y)$. The transition relation $\rightsquigarrow$ is defined as the union of transition relations $\rightsquigarrow_l$ with $l \in L$. The labels indicate the action taken (1st letter), to which formula or variable set it is applied (2nd letter), whether it is triggered by a satisfying or falsifying assignment (3rd letter) and whether a blocking clause is learned or flipping is applied (4th letter). The rules are listed in Figure 11.2.

Terminate search. The search terminates as soon as I either satisfies or falsifies either P or N and the relevant search space has been traversed exhaustively. If I falsifies P, $\pi(I,X)$ can not be extended to a model of $\exists Y \,.\, F(X,Y)$, and M remains unaltered (EP0). Requiring that no decision literals are left on the trail ensures that no models are missed due to a "wrong" assignment of variables in S. If I either satisfies P or falsifies N, $\pi(I,X)$ can be extended to $2^{|X-I|}$ models of $\exists Y \,.\, F(X,Y)$, EP0 : $(P,\, N,\, I,\, M) \rightsquigarrow_{\mathsf{EP0}} M$ if $\emptyset \in P|_I$ and $\mathsf{decs}(I) = \emptyset$

EP1: $(P,\, N,\, I,\, M) \rightsquigarrow_{\mathsf{EP1}} M + 2^{|X-I|}$ if $P|_I = \emptyset$ and $\mathsf{var}(\mathsf{decs}(I)) \cap X = \emptyset$

EN0: $(P,\, N,\, I,\, M) \rightsquigarrow_{\mathsf{EN0}} M + 2^{|X-I|}$ if $\emptyset \in N|_I$ and $\mathsf{var}(\mathsf{decs}(I)) \cap X = \emptyset$

BP0F: $(P,\, N,\, I\ell^d I',\, M) \rightsquigarrow_{\mathsf{BP0F}} (P,\, N,\, I\bar{\ell}^{f(m')},\, M)$ if $\emptyset \in P|_{I\ell I'}$ and $\mathsf{var}(\mathsf{decs}(I')) = \emptyset$ and $m' = \sum \{m \mid \ell^{f(m)} \in I'\}$

JP0: $(P,\, N,\, II',\, M) \rightsquigarrow_{\mathsf{JP0}} (P \wedge C^r,\, N,\, I\ell',\, M - m')$ if $\emptyset \in P|_{II'}$ and $P \models C$ and $C|_I = \{\ell'\}$ and $m' = \sum \{m \mid \ell^{f(m)} \in I'\}$

BN0F: $(P,\, N,\, I\ell^d I',\, M) \rightsquigarrow_{\mathsf{BN0F}} (P,\, N,\, I\bar{\ell}^{f(m'+m'')},\, M + m'')$ if $\emptyset \in N|_{I\ell I'}$ and $\mathsf{var}(\ell) \in X$ and $\mathsf{var}(\mathsf{decs}(I')) \cap X = \emptyset$ and $m' = \sum \{m \mid \ell^{f(m)} \in I'\}$ and $m'' = 2^{|X - I\ell I'|}$

BN0L: $(P,\, N,\, I\ell^d I',\, M) \rightsquigarrow_{\mathsf{BN0L}} (P \wedge D,\, N,\, I\bar{\ell},\, M + m'')$ if $\emptyset \in N|_{I\ell I'}$ and $\mathsf{var}(\ell) \in X$ and $\mathsf{var}(\mathsf{decs}(I')) \cap X = \emptyset$ and $m'' = 2^{|X - I\ell I'|}$ and $D = \pi(\neg\mathsf{decs}(I\ell), X)$

BP1F: $(P,\, N,\, I\ell^d I',\, M) \rightsquigarrow_{\mathsf{BP1F}} (P,\, N,\, I\bar{\ell}^{f(m'+m'')},\, M + m'')$ if $P|_{I\ell I'} = \emptyset$ and $\mathsf{var}(\ell) \in X$ and $\mathsf{var}(\mathsf{decs}(I')) \cap X = \emptyset$ and $m' = \sum \{m \mid \ell^{f(m)} \in I'\}$ and $m'' = 2^{|X - I\ell I'|}$

BP1L: $(P,\, N,\, I\ell^d I',\, M) \rightsquigarrow_{\mathsf{BP1L}} (P \wedge D,\, N,\, I\bar{\ell},\, M + m'')$ if $P|_{I\ell I'} = \emptyset$ and $\mathsf{var}(\ell) \in X$ and $\mathsf{var}(\mathsf{decs}(I')) \cap X = \emptyset$ and $m'' = 2^{|X - I\ell I'|}$ and $D = \pi(\neg\mathsf{decs}(I\ell), X)$

(a) End and backtracking rules.

Figure 11.2: The complete set of rules of our framework, where $P(X,Y,S)$ and $N(X,Y,T)$ form a combined formula pair of $F(X,Y)$, and I denotes the trail over variables $X \cup Y \cup S \cup T$. The rules cover termination (rule name starting with E), chronological and non-chronological backtracking (B and J), decisions (D) and unit propagation (U) as well as clause forgetting (F). They may be applied either to P or N (P or N) and triggered by a falsifying (0) or satisfying (1) assignment. Letters X, Y, S and T denote the variable sets the rules are applied to (X, Y, S and T). Finally, either a blocking clause may be learned (L) or flipping is applied (F). Blocking clauses are added to P only and thus fail to prevent multiple model counts if a model is found due to a conflict in N. We therefore disallow the combination of blocking clauses and dual reasoning. Discounted models might not be detected again if they are subsumed by a blocking clause recorded previously. We therefore also disallow the combination of discounting and blocking clauses. For more details including examples we refer to paragraphs *Blocking clauses in dual mode* and *Combining blocking clauses and discounting* in Section 11.4.

DX:	$(P, N, I, M) \rightsquigarrow_{\mathsf{DX}} (P, N, I\ell^d, M)$ if $\emptyset \notin (P \wedge N)\vert_I$ and $\mathsf{units}((P \wedge N)\vert_I) = \emptyset$ and $\mathsf{var}(\ell) \in X - I$
DYS:	$(P, N, I, M) \rightsquigarrow_{\mathsf{DYS}} (P, N, I\ell^d, M)$ if $\emptyset \notin (P \wedge N)\vert_I$ and $\mathsf{units}((P \wedge N)\vert_I) = \emptyset$ and $\mathsf{var}(\ell) \in (Y \cup S) - I$ and $X - I = \emptyset$
UP:	$(P, N, I, M) \rightsquigarrow_{\mathsf{UP}} (P, N, I\ell, M)$ if $\{\ell\} \in P\vert_I$
UNXY:	$(P, N, I, M) \rightsquigarrow_{\mathsf{UNXY}} (P, N, I\overline{\ell}^d, M)$ if $\{\ell\} \in N\vert_I$ and $\mathsf{var}(\ell) \in X \cup Y$ and $\emptyset \notin P\vert_I$ and $\mathsf{units}(P\vert_I) = \emptyset$
UNT:	$(P, N, I, M) \rightsquigarrow_{\mathsf{UNT}} (P, N, I\ell, M)$ if $\{\ell\} \in N\vert_I$ and $\mathsf{var}(\ell) \in T$ and $\emptyset \notin P\vert_I$ and $\mathsf{units}(P\vert_I) = \emptyset$
FP:	$(P \wedge C^r, N, I, M) \rightsquigarrow_{\mathsf{FP}} (P, N, I, M)$ if $\emptyset \notin P\vert_I$

(b) Decision and unit propagation and forgetting rules.

Figure 11.2: The complete set of rules of our framework (cont.).

and M is updated accordingly (EP1 and EN0). Requiring that no relevant decision literal is left on the trail is sufficient since in the presence of irrelevant or primal decision literals all relevant variables are assigned. Flipping an irrelevant or primal decision literal therefore would yield redundant models w.r.t. projection onto X.

Backtracking. If the partial interpretation represented by the trail falsifies P, the solver may backtrack chronologically and turn the last decision literal ℓ^d into a flipped decision literal $\overline{\ell}^{f(m)}$ where m equals the number of models detected at decision level $> \mathsf{dl}(\ell)$. This model count is obtained by summing up the model counts assigned to the flipped decision literals with decision level higher than $\mathsf{dl}(\ell)$ while M remains unaltered (BP0F). Alternatively, a redundant clause may be learned which becomes unit for I, the solver backtracks non-chronologically and propagates this new unit literal. Backjumping involves discarding the models found in I', hence their count is subtracted from M to prevent multiple counts when they are found again (JP0). If the partial interpretation represented by the trail falsifies N, its projection onto X can be extended to a model of F, and M is incremented by the number of total models projected onto X represented by the trail. Flipping the last irrelevant or primal decision literal would yield redundant models w.r.t. projection onto X. Therefore, when backtracking chronologically, the solver turns the last relevant decision literal into a flipped decision literal and assigns it the sum of the number of models represented by the actual trail projected onto X and all models detected at decision levels $> \mathsf{dl}(\ell)$ (BN0F). Alternatively, a blocking clause is added to P (BN0L). If $I\ell^d I'$ satisfies P, $\pi(I\ell I', X)$ can be extended to $m'' = 2^{|X - I\ell I'|}$ models of $\exists Y . F(X, Y)$, and M is incremented by m''. As discussed above, the last relevant decision literal ℓ must be considered. It may be turned into a flipped decision literal and assigned the sum of m'' and all number of models detected at decision level $> \mathsf{dl}(\ell)$ (BP1F). Alternatively, a blocking clause may be added to P (BP1L).

Decisions. $P|_I$ and $N|_I$ contain neither a unit nor the empty clause. Relevant input variables are prioritized (DX) over irrelevant input and primal variables (DYS). Assigning primal variables before irrelevant input variables might result in a conflict in P but has no effect on N and hence does not affect the model count. In this case, the duality property might not hold for the residuals as discussed in Section 11.3.

Unit propagation. Literals are propagated in both P and N. Unit propagation in P is prioritized over unit propagation in N and is executed as in SAT (UP). For unit propagation in N, two cases need to be considered. If $\mathsf{var}(\ell) \in X \cup Y$, I is extended by $\bar{\ell}^d$ to enforce backtracking due to a conflict in $N|_{I\bar{\ell}}$ in the next step (UNXY). Otherwise, it is propagated (UNT).

Forgetting redundant clauses. Deletion of redundant clauses is equivalence-preserving, hence redundant clauses may be removed anytime (FP) assuming P is conflict-free under I.

Blocking clauses in dual mode. Blocking clauses shall impede the multiple detection of models. Since they are added exclusively to P, this fails in the dual setting if a trail falsifies N. Consider $F(X, \emptyset) = (\bar{1} \vee 2) \wedge (1 \vee 2)$ over $X = \{1, 2, 3, 4\}$ and $Y = \emptyset$. We define $P = F$ and $N = (\bar{5} \vee 1) \wedge (\bar{5} \vee \bar{2}) \wedge (\bar{6} \vee \bar{1}) \wedge (\bar{6} \vee \bar{2}) \wedge (5 \vee 6)$, hence $S = \emptyset$ and $T = \{5, 6\}$. After deciding 4, 3 and 2 (DX), $\bar{5}$ and $\bar{6}$ are propagated in N (UNT) resulting in a conflict in $N|_I$ with $I = (4^d, 3^d, 2^d, \bar{5}, \bar{6})$ and $\pi(I, X) = (4, 3, 2)$. The solver adds the blocking clause $(\bar{2} \vee \bar{3} \vee \bar{4})$ to P, sets $M = 2$ and backtracks chronologically (BN0L). After propagating $\bar{1}$ (UP), P is falsified. JP0 is applied yielding the redundant unit clause (2), and the solver jumps back to decision level 0. It propagates 2 UP), $\bar{5}$ and $\bar{6}$ (UNT) resulting in $I = (2, \bar{5}, \bar{6})$ which falsifies N. The partial model $\pi(I, X) = (2)$ represents $2^3 = 8$ total models of F. Note that this model subsumes the one found previously. **In our framework, we therefore have to disallow the combination of blocking clauses and dual reasoning.** This means that "L" rules can not be combined with "N" rules. Thus only "F" rules are allowed if we insist on using "N" rules, which is the default in our implementation.

A workaround would be the following: If a blocking clause is added to P, either its negation is added disjunctively to N or, whenever a conflict in N occurs, it must be ensured that none of the blocking clauses is falsified by the current trail. In the first case, N is not a CNF anymore, and the rules and preconditions involving N and the whole search procedure need to be adapted accordingly. In the second case we need to keep track of the blocking clauses instead, e.g., in a CNF R (*refutations*), and check whether R is satisfied prior to count a model whenever N is falsified by I.

Combining blocking clauses and discounting. If previously found models are *discounted* upon backjumping and learning a blocking clause, they may be lost. The problem in this case arises if these models are blocked by the learned clause. Consider $F = (1 \vee \bar{2}) \wedge (1 \vee \bar{3}) \wedge (\bar{1} \vee \bar{2})$. $P = F$ and $N = (\bar{6} \vee \bar{1}) \wedge (\bar{6} \vee 2) \wedge (\bar{7} \vee \bar{1}) \wedge (\bar{7} \vee 3) \wedge (\bar{8} \vee 1) \wedge (\bar{8} \vee 2) \wedge (6 \vee 7 \vee 8)$ with $X = \{1, 2, 3, 4, 5\}$, $Y = \emptyset$, $S = \emptyset$, and $T = \{6, 7, 8\}$. After deciding 5, 4, and 3, 1 and $\bar{2}$ are propagated which leads to the detection of model $m_1 = (1, \bar{2}, 3, 4, 5)$. The last decision is flipped (BP1F) and after propagating $\bar{7}$, deciding $\bar{2}$ and propagating $\bar{6}$ and $\bar{8}$, a second model $m_2 = (\bar{2}, \bar{3}, 4, 5)$ is found. The solver backtracks and flips the last decision (BN0F). At a later stage, BP0F is executed, and $I = (5^d, \bar{4}^{f(3)})$. The model count associated with the flipped decision literal $\bar{4}^{f(3)}$ refers to models m_1 and m_2. Later on, a third model $m_3 = (1, \overline{2}, \overline{4}, 5)$ is detected and (BN0F) executed. After one more propagation step, model $m_4 = (\overline{1}, \overline{2}, \overline{3}, \overline{4}, 5)$ is detected and a blocking clause $b = (2 \vee \overline{5})$ learned (BP1L). Due to the application of JP0 in the further procedure, the solver jumps back to level 0, i.e., past the flipped decision literal $\overline{4}^{f(3)}$. It discounts models m_1 and m_2, i.e., 3 total models. Since these models are blocked by b, they can not be found again. In fact, during the further execution the models $m_5 = (1, \overline{2}, \overline{5})$ and $m_6 = (\overline{1}, \overline{2}, \overline{3}, \overline{5})$ are found before terminating with EN0, and the solver returns $M = 9$ instead of $M = 12$. Combining the use of blocking clauses with discounting therefore conditions to discount only models which are not blocked by the learned clause and to add blocking clauses for these models as soon as the respective flipped decision literal is removed from I. An obvious drawback of this method is that these checks are expensive since the number of models may be exponential in the number of relevant variables. **Thus our framework also disallows the combination of discounting and blocking clauses**. This means that as soon we have discounting enabled, actually backjumping (JP0) with $m' \neq 0$, we can not use blocking clauses ("L" rules). This provides another reason to disable "L" rules in our implementation by default, even though for testing purposes blocking clauses can be enabled (if dual reasoning and discounting are disabled).

The demonstrated issues concerning the use of blocking clauses did not use irrelevant variables, but the argument can easily be lifted to the case where $Y \neq \emptyset$.

11.5 IMPLEMENTATION

We have implemented the calculus of Section 11.4 in our new tool Dualiza implemented from scratch in C available at `http://fmv.jku.at/dualiza`. The tool counts and prints the number of models, optionally projected on a set of relevant variables. It can also act as a simple SAT solver, or enumerate all (projected) partial models, i.e., compiles a disjunctive representation of all (projected) models.

We carefully tested our tool against state-of-the-art SAT solvers and the model counter sharpSAT [189] using a regression test suite which comes with the tool but also using fuzz testing [35]. Beside the main CNF-based back-end implementing of our calculus, we also provide our own BDD engine, which obviously only scales for small formulae, but is useful to test projected model counting.

The front-end of Dualiza reads Boolean formulae in a simple format, circuits in AIGER format [18], or CNF in DIMACS format, and encodes them into CNF using the Plaisted-Greenbaum transformation [164] after flattening the internal circuit representation. As future work we plan to further optimize the internal circuit representation. The same procedure is applied to the negated formula to obtain a dual representation.

The resulting primal and dual CNFs (P and N) are individually preprocessed by bounded variable elimination [65], which is restricted to only eliminate irrelevant variables ($S \cup T$). We will add more preprocessing and in particular more dedicated preprocessing techniques such as those from [115] in the future.

The core engine of Dualiza uses standard data structures and algorithms implemented in state-of-the-art CDCL solvers, including watched literals, activity-based decision heuristics (actually VMTF), frequent garbage collection of learned clauses, and also frequent restarts (as long as no variable is flipped). An important optimization is to allow picking decisions in an arbitrary order until a first model is found. Then we restart without counting this first model and afterwards enforce splitting on relevant variables first. Due to phase saving this first model is found again instantly.

The source code marks the implementation of every rule of our calculus through macros (“`RULE`”), which allows to gather statistics about their application and also can be used to produce traces of the application of our calculus on concrete examples (“`dualiza -l | grep RULE`”).

11.6 experiments

Consider again the example in Figure 11.1. The solver trace (enabled with “`-l`”) shows the decision level (number of decisions plus flipped decisions) in the 3rd column, followed by the name of the rule and a running counter for each rule. The last column gives the (encoded) flipped or added literal.

The simplest interesting example is a CNF formula consisting of a single clause of n variables. In dual mode such a formula is solved by dual propagation alone. Consider for $n = 4$ the simple Boolean formula listed on the left hand side of Figure 11.1 in the formula input format of Dualiza. In dual mode Dualiza can enumerate and count the number of such a formula instantly, i.e., for $n = 10000$ the number of models $2^{10000} - 1$ is computed in 0.16 seconds. Disabling dual mode (“`-no-dual`”) leads to exponential run-times in n, since our implementation requires that all variables are assigned before detecting that P is empty (rules BP1F and BP1L), as it is common in SAT solvers. Already for $n = 30$ it takes more than 215 seconds when only applying our flipping rules and the procedure does not terminate within one hour when only using blocking clauses (even with the subsumption check described in Section 11.8 enabled).

For component-based model counting as implemented in the state-of-the-art exact model counter sharpSAT [189] such single clause instances are trivial too (10.58 seconds for $n = 10000$), since for each decision in one branch the formula is satisfied and in the other branch the clause is reduced in size. Actually such a solver could in principle instantly determine the number of models as soon as a component consists of a single clause.

Thus in order to show the orthogonal strength of our approach compared to component-based model counting, we also experimented with formulae where splitting on variables does not partition the formula into disconnected components. Consider for $n = 4$ the following formula

```
(x1 | x2 | x3 | x4) |
(x5 = x2 ^ x3 ^ x4) |
(x6 = x1 ^ x3 ^ x4) |
(x7 = x1 ^ x2 ^ x4) |
(x8 = x1 ^ x2 ^ x3)
```

The first row is similar to the previous example. The other equalities evaluate to true if the new variable on the left is identical to the parity (“ˆ” denotes XOR) of different sets of three variables. The number of models is $2^{2n} - 1$, e.g., only the assignment where the first n variables are false and the second half are true is not a model. Our implementation takes 244.37 seconds to compute $2^{10000} - 1$ models for $n = 5000$, while sharpSAT shows exponential behavior and can only compute the model count up to $n = 21$ within one hour.

In a related experiment we took the 127 satisfiable CNF benchmarks from the main track of the SAT Competition 2017, for which at least one solver produced a correct witness during the competition within 5000 seconds (on a slightly faster machine). Our tool finds at least one model for 60 benchmarks within the same time limit (best solver in the competition 102), and provides an exact model count for 22 in primal mode and 16 in dual mode. This shows that dual reasoning without projection is in our current implementation not really effective for benchmarks in CNF. However, sharpSAT could only provide an exact model count for one benchmark ("9_38"), runs out of memory on 10 and actually also produced two discrepancies (claimed that both "ak128modbtbg2msisc" and "UCG-20-10p1" have no solutions).

The most recent paper on *projected* model counting [10] used random benchmarks and a set of planning benchmarks, which are only available in CNF. It turns out that our tool is not really competitive on these instances without component reasoning. Due to the non-structural CNF input format, dual propagation is not effective and our tool in essence enumerates all (projected) models explicitly. We are exploring other potential sources of benchmarks for projected model counting, including AIGER circuits from hardware model checking.

Our experiments used an Intel Xeon E5-2620 v4 CPU at 2.10GHz (turbo-mode disabled) with memory limit of 31 GB.

11.7 RELATED WORK

Due to its wide practical applicability, research on propositional model counting exhibits an impressive diversity, as demonstrated by the following list of related work.

Exact model counting. A widely used paradigm is that of splitting the formula into subformulae called *components* with disjoint variable sets which are then processed separately [15]. The authors also demonstrate the need for so-called *good learning* in contrast to *nogood learning* realized by CDCL. Combining component caching and clause learning resulted in a significant performance gain [173]. In sharpSAT [189], *implicit BCP* and an improved component caching additionally reduces search space and cache size. The caching scheme was adapted to support parallelization [37] and distributivity [38]. For a survey on exact model counting algorithms we refer to [144].

Approximate model counting. In some applications, such as probabilistic reasoning, an approximated model count would suffice. Let us mention two paradigms originating from probability theory. The first is based on sampling [45, 86], while in the second (short) XOR constraints are added to the formula until it becomes unsatisfiable [1, 88]. Exact and approximate model counting algorithms are compared in [89].

Alternative methods. All mentioned exact model counters so far are based on the DPLL algorithm [60] or an enhancement thereof. An alternative approach consists in compiling the formula into a language in which model counting is tractable [16, 59] applied in, e.g., conformant planning [162]. In theory structure-based approaches can for instance also make use of the community structure of the formula [81] or the structure of the hypergraph associated with the formula [41, 42].

11.8 CONCLUSION

We have presented a dual procedure for projected model counting taking into account the formula as well as its negation. We devised an efficient good learning mechanism based on the detection of partial models, which is exempt from satisfiability checks and clause watching. This method enables the pruning of a potentially large portion of the search space. Formulae with large satisfying subspaces of the search space benefit most. For these, the learned goods tend to be short. We introduced the concepts of flipping and discounting to remember the models found at the actual decision level. This allows us to jump back over branches in which models were found by simply subtracting their count from the number of models found so far. Completeness of CDCL guarantees that these models are found again at a later stage. Flipping and discounting render the use of blocking clauses superfluous and thus prevent an additional growth of the formula.

We have preliminary ideas on how to handle backjumping and redundant dual clause learning for dual conflicts. This has the potential to shrink partial models substantially, particularly in combination with discounting. In the current calculus this situation is addressed by backtracking (BN0F and BN0L).

Our method prioritizes decisions of the relevant variables over the irrelevant variables. By adopting the search strategy utilized in [84], we might be able to relax this restriction. Note that with every relevant flipped decision the models grow larger. We plan to investigate whether overall backjumping upon a conflict in N can compensate restrictions imposed by our branching heuristics. Moreover, the algorithm presented in [84] allows to remove blocking clauses eagerly and explicitly as soon they are not needed anymore. This is not captured in our calculus yet. In our implementation we provide a poor-man's version simulating this approach partially, by trying to subsume previous blocking clauses by a new blocking clause.

Another important future extension of our calculus is to capture component reasoning, which can give exponential speed-ups orthogonal to what can be achieved by dual reasoning.

Beside the more technical challenge to improve the CNF encoding through circuit optimizations and extending and applying more preprocessing techniques to the generated CNFs individually, we also want to explore more general preprocessing techniques which jointly work on the dual representation, taking advantage of the duality property.

Finally, Dualiza makes it easy to apply model counting in various domains. This in turn will allow the research community to obtain interesting model counting benchmarks and in general encourages more research in model counting.

12

DISCUSSION OF PAPER 2

The paper is summarized and the main contributions are pointed out and precised (Section 12.1). More details and examples are provided concerning the decision strategy and the flipping and discounting mechanism, which due to space requirements were only shortly touched upon in the paper (Section 12.2 and Section 12.3), as well as unit propagation in the dual formula (Section 12.5). After the publication of the paper, we have worked on understanding how dual conflicts can be handled, a topic listed in the paper as future research direction. It turned out that conflict-driven backjumping in the dual formula might result in the loss of models and can therefore only be adopted in the primal formula (Section 12.4). Using blocking clauses (L rules) in dual mode or in combination with flipping and discounting (F rules) may lead to an incorrect result (Section 12.6). Finally, we have repeated the experiments reported in the paper and run the same experiments on model counters introduced after the publication of our paper. The results are presented in Section 12.7.

12.1 MAIN CONTRIBUTIONS

We extended our calculus Abstract Dual #DPLL [24] by projection capabilities and by conflict-driven clause learning (CDCL) with non-chronological backjumping. The former facilitates the processing of formulae in which not all variables are relevant for a given task and of formulae containing auxiliary variables, while the latter allows for taking advantage of learning from conflicts.

Our dual model counter Dualiza takes as argument an arbitrary formula or a circuit $F(X,Y)$ defined over the set of relevant input variables X and the set of irrelevant input variables Y, where $X \cap Y = \emptyset$. It transforms $F(X,Y)$ and $\neg F(X,Y)$ into CNFs $P(X,Y,S)$ and $N(X,Y,T)$ with auxiliary variables S and T by means of the Tseitin encoding [192], but the Plaisted-Greenbaum [164] transformation could be adopted as well. As we have seen in Chapter 6, the Tseitin encoding preserves the model count of F, because it replaces a subformula G of F with a fresh variable u and adds its definition $(u \leftrightarrow G)$ to F, hence in all total models of $\mathsf{tseitin}(F)$, both u and G evaluate to the same truth value. In contrast, in the Plaisted-Greenbaum transformation either $(u \rightarrow G)$ or $(u \leftarrow G)$ is added to F, hence the Plaisted-Greenbaum transformation does not preserve the model count. However, the total models of P projected onto $X \cup Y$ are exactly the total models of F projected onto $X \cup Y$, and the Plaisted-Greenbaum transformation preserves the model count under projection.

In Abstract Dual #DPLL, the formulae P and N are defined over the same set of variables. This is not the case anymore for $P(X,Y,S)$ and $N(X,Y,T)$, since S

and T are disjoint. Consequently, the residuals of P and N under a trail are not correlated, either. These observations are taken into account by partitioning a total assignment $\sigma\colon X \cup Y \cup S \cup T \mapsto \mathbb{B}$ into the relevant input assignment $\sigma_X\colon X \mapsto \mathbb{B}$ and the irrelevant input assignment $\sigma_Y\colon Y \mapsto \mathbb{B}$ as well as the primal assignment $\sigma_S\colon S \mapsto \mathbb{B}$ and the dual assignment $\sigma_T\colon T \mapsto \mathbb{B}$, and by the dual formula representation of the input formula, which for $F(X, Y)$ consists of $P(X, Y, S)$ and $N(X, Y, T)$. The dual formula representation provides the basis for our definition of the duality property capturing the relation between P and N. Basically, the duality property says that whenever a total assignment to the variables in $X \cup Y$ can be extended to an assignment satisfying P, it can not be extended to an assignment satisfying N and vice versa, and it holds for residuals as well, provided relevant variables are decided before irrelevant and primal ones and dual variables are never decided, as explained in further down. The dual representation enables us to find partial models without implementing clause watching mechanisms or executing satisfiability checks. Particularly the latter is expensive for large formulae. It also impacts the order in which variables need be decided as well as the termination conditions. In the paper, these conditions were only shortly touched upon, and details are provided in Section 12.2.

In order to avoid finding models multiple times after conflict-driven backtracking, we introduce the concepts of flipping and discounting. In the paper, this technique was only shortly sketched and introduced by the corresponding rules. We provide an example in Section 12.3. The investigation of the combination of dual reasoning, flipping and discounting, and good learning provided us with a further understanding of our dual approach.

We extended our tool DualCountPro by the new rules. It turned out to be very helpful in checking our rules. The execution trace and running time in the dual mode is mostly significantly shorter than in primal, i. e., non-dual, mode. This contrasts our observations in the new tool Dualiza implemented from scratch in C. We present DualCountPro in Chapter 13.

12.2 DECISION STRATEGY

Counting or enumerating models under projection involves exploring the relevant search space, which consists of all possible assignments to the relevant input variables σ_X. Therefore, if a model has been found, a relevant decision need be flipped to ensure that an unexplored relevant input assignment is obtained. And to ensure that all possible relevant assignments are tested, relevant input variables must be decided before irrelevant ones: if the most recent relevant decision on the trail is flipped and a conflict in P is obtained due to some irrelevant or primal assignment, this particular relevant assignment might not be encountered again. If its projection onto $X \cup Y$ could be extended to a model of $\exists S \,.\, P(X, Y, S)$, the corresponding model is missed. The following example clarifies this idea.

Example 12.1 (Deciding irrelevant inputs before relevant inputs)**.** *Consider*

$$\begin{aligned} P(X,Y,S) = {} & \underbrace{(a \lor b \lor c \lor s_1)}_{P_1} \land \underbrace{(a \lor b \lor c \lor \neg s_1)}_{P_2} \land \\ & \underbrace{(a \lor b \lor \neg c \lor s_1)}_{P_3} \land \underbrace{(a \lor b \lor \neg c \lor \neg s_1)}_{P_4} \land \\ & \underbrace{(\neg a \lor \neg b \lor c \lor s_1)}_{P_5} \land \underbrace{(\neg a \lor \neg b \lor c \lor \neg s_1)}_{P_6} \land \\ & \underbrace{(\neg a \lor \neg b \lor \neg c \lor s_1)}_{P_7} \land \underbrace{(\neg a \lor \neg b \lor \neg c \lor \neg s_1)}_{P_8} \land \\ & \underbrace{(c \lor s_1 \lor s_2)}_{P_9} \land \underbrace{(c \lor s_1 \lor \neg s_2)}_{P_{10}} \end{aligned}$$

defined over the set of relevant input variables $X = \{a\}$, the set of irrelevant input variables $Y = \{b, c\}$, and the set of primal variables $S = \{s_1, s_2\}$, and

$$\begin{aligned} N(X,Y,T) = {} & \underbrace{(a \lor \neg b \lor c \lor t_1)}_{N_1} \land \underbrace{(\neg a \lor b \lor c \lor \neg t_1)}_{N_2} \land \\ & \underbrace{(a \lor \neg b \lor \neg c \lor t_1)}_{N_3} \land \underbrace{(\neg a \lor b \lor \neg c \lor \neg t_1)}_{N_4} \land \\ & \underbrace{(\neg a \lor b \lor c \lor t_1)}_{N_5} \land \underbrace{(a \lor \neg b \lor c \lor \neg t_1)}_{N_6} \land \\ & \underbrace{(\neg a \lor b \lor \neg c \lor t_1)}_{N_7} \land \underbrace{(a \lor \neg b \lor \neg c \lor \neg t_1)}_{N_8} \land \\ & \underbrace{(\neg c \lor t_1 \lor t_2)}_{N_9} \land \underbrace{(\neg c \lor t_1 \lor \neg t_2)}_{N_{10}} \end{aligned}$$

defined over the same sets of relevant and irrelevant variables X and Y, and the set of dual variables $T = \{t_1, t_2\}$. Notice that P and N comply with the duality property (Definition 11.2): it holds that $\forall X, Y \,.\, ((\exists S \,.\, P(X,Y,S)) \oplus (\exists T \,.\, N(X,Y,T)))$, and whenever for a total input assignment $\sigma_X \cup \sigma_Y$ there exists a total primal assignment σ_S such that $(\sigma_X \cup \sigma_Y \cup \sigma_S)(P(X,Y,S)) = 1$, for all possible total assignments σ_T we have that $(\sigma_X \cup \sigma_Y \cup \sigma_T)(N(X,Y,T)) = 0$, and vice versa. In fact, the models of P projected onto $X \cup Y$ are $a \neg b c$, $a \neg b \neg c$, $\neg a b c$, and $\neg a b \neg c$, and the models of N projected onto $X \cup Y$ are $a b c$, $a b \neg c$, $\neg a \neg b c$, and $\neg a \neg b \neg c$. We are interested in the models of P projected onto X, which are given by $\mathsf{models}(\exists Y, S \,.\, P(X,Y,S)) = \{a, \neg a\}$.

The trail $\neg b^d c^d a^d$ satisfies P, and the model $m_1 = a$ is recorded. Flipping a^d and propagating s_1 with reason P_3 results in a conflict in P, since P_4 becomes empty. The resulting trail $\neg b^d c^d \neg a$ contains no relevant decision, and the computation terminates returning a and missing $\neg a$.

In our rules, irrelevant and primal variables are treated equally with respect to the decision strategy, i. e., after all variables in X are assigned, variables in both Y and S may be decided. The projection of the resulting trail onto X is not affected by the decision whether primal or irrelevant input variables are decided first. Furthermore, only relevant decisions are flipped after finding a model, hence only pairwise contradicting models are found, and since relevant decisions are prioritized, all models projected onto the relevant input variables X are found.

Since deciding primal variables is permitted, a primal variable might be assigned a value such that the resulting trail I falsifies P, even if its projection onto $X \cup Y$ is a model of $\exists S \,.\, P(X, Y, S)$. In this case, the trail I can not be extended to a model of N, either, and the duality property does not hold hold anymore. However, this has no effect on the model count: to ensure that all possible assignments are tested, after a conflict the most recent decision need be flipped, be it relevant or irrelevant or primal, and a satisfying assignment differing only in primal variables from I will be found. If instead the duality property is violated due to the decision of a dual variable and a conflict in N occurs, the projection of the current trail onto $X \cup Y$ is considered a model of $\exists S \,.\, P(X, Y, S)$, and a countermodel of $\exists S \,.\, P(X, Y, S)$ is counted or enumerated, as shown by an example.

Example 12.2 (Violation of the duality property)**.** *Consider again Example 12.1 and let the current trail be $I = a^d \neg b^d \neg {s_1}^d \neg c^d {s_2}^{P_9}$. The clause P_{10} is empty under I, and therefore $P|_I = 0$. But I can not be extended to a model of N, either. After flipping the most recent decision $\neg c^d$, the resulting trail $a^d \neg b^d \neg {s_1}^d c$ satisfies P. Its projection onto X is a, which is exactly I projected onto X. The most recent relevant decision is flipped followed by deciding again $\neg s_1$ and propagating c with reason P_5, after which P_7 becomes empty. The current trail is $a^d b \neg {s_1}^d c^{P_5}$. The most recent decision $\neg {s_1}^d$ is flipped and the literal c is propagated with reason P_6 resulting in P_8 becoming empty. After backtracking chronologically and deciding b and c, the trail $\neg a b^d c^d$ satisfies P. Its projection onto X is $\neg a b$. The trail contains no relevant decision and the computation terminates.*

Now let our trail be $J = a^d b^d \neg {t_1}^d \neg c^d {t_2}^{N_9}$. The clause N_{10} is empty under J, and J can not be extended to a model of P, either. However, the preconditions of rules BN0F *and* BN0L *are met, and $a b$ is counted, which clearly is wrong.*

We therefore prioritize the deciding relevant input variables and never decide dual variables. This decision strategy is reflected in the rules depicted in Figure 11.2b. It is still possible to count the models of P projected onto X without considering N. Instead, the computation might not be possible considering only N, namely in the case where all input variables are assigned and unit propagation has been performed but the residual of N under the resulting trail is still undefined.

The following invariant stated in Equation 11.5 provides the basis for our dual model counting method and holds anytime under this decision strategy:

$$\forall X, Y \,.\, ((\neg \exists\, T \,.\, N(X, Y, T)|_I) \rightarrow (\exists\, S \,.\, P(X, Y, S)|_I))$$

First, since $P(X, Y, S)$ and $N(X, Y, S)$ comply with the duality property stated in Equation 11.3, a total assignment $\sigma_X \cup \sigma_Y$ which can not be extended to a model of N can be extended to a model of P. Second, dual variables are never decided. They are therefore either assigned according to their definition or not assigned at all, and it can not happen that the duality property is violated due to a "wrong" assignment to a dual variable. From this it follows that whenever a conflict in N occurs, the projection of the trail onto $X \cup Y$ satisfies $\exists S \,.\, P(X, Y, S)$, which is reflected in the rules BN0F and BN0L of our calculus shown in Figure 11.2a.

12.3 FLIPPING AND DISCOUNTING BY AN EXAMPLE

Consider again Example 12.1 but with $X = \{a, b, c\}$ and $Y = \emptyset$. The following discussion can readily be lifted to the case where $Y \neq \emptyset$. We are interested in the number of total models of $P(X, Y, S)$ projected onto X, which is given by $|\mathsf{models}(\exists Y, S \,.\, P(X, Y, S))| = |\{a \neg b \neg c, a \neg b c, \neg a b c, \neg a b \neg c\}| = 4$.

Table 12.1: Flipping and discounting example.

STEP	I	RULE	J	MODEL	MC
1	$a^d \neg b^d \neg c^d t_1{}^{N_5}$	BN0F	$a^d \neg b^d c^{f(1)}$	$a \neg b \neg c$	1
2	$a^d \neg b^d c^{f(1)}$	BP1F	$a^d b^{f(2)}$	$a \neg b c$	2
3	$a^d b^{f(2)} \neg c^d s_1{}^{P_5}$	BP0F	$a^d b^{f(2)} c^{f(0)}$		2
4	$a^d b^{f(2)} c^{f(0)} s_1{}^{P_7}$	BP0F	$\neg a^{f(2)}$		2
5	$\neg a^{f(2)} b^d c^d$	BP1F	$\neg a^{f(2)} b^d \neg c^{f(1)}$	$\neg a b c$	3
6	$\neg a^{f(2)} b^d \neg c^{f(1)} t_1{}^{N_1}$	BN0F	$\neg a^{f(2)} \neg b^{f(2)}$	$\neg a b \neg c$	4
7	$\neg a^{f(2)} \neg b^{f(2)} c^d s_1{}^{P_3}$	BP0F	$\neg a^{f(2)} \neg b^{f(2)} \neg c^{f(0)}$		4
8	$\neg a^{f(2)} \neg b^{f(2)} \neg c^{f(0)} s_1{}^{P_1}$	EP0	$\neg a^{f(2)} \neg b^{f(2)} \neg c^{f(0)} s_1{}^{P_1}$		4

For the sake of conciseness, the discussion is restricted to the interesting situations, which are those in which the trail either satisfies or falsifies P or N. The intermediate execution steps can be retraced based on the trail. The execution trace is shown in Table 12.1. The first column denotes the line number. The columns I and J refer to the trail before and after the execution of the rule listed in the third column. The fifth and sixth column represent the model found in this step and the number of models found so far and initialized with zero, respectively. In the rest of this subsection, the executions steps are discussed one by one.

Step 1: We have $N_2|_I = 0$, hence the projection of I onto X, $\pi(I, X) = a \neg b \neg c$, is a model of $\exists Y, S \,.\, P(X, Y, S)$. It is a total model, and the model count MC is incremented by one. The model counter backtracks chronologically applying the rule BN0F. The decision $\neg c^d$ is flipped and marked as a flipped decision literal with the number of models found becoming $c^{f(1)}$.

Step 2: The resulting trail satisfies P, and chronological backtracking occurs by means of the rule BP1F. The model found is total under projection onto X. The variable c is unassigned, the model count associated with it is summed up to the number of models represented by I, and the decision $\neg b^d$ becomes $b^{f(2)}$. The number of models found so far is incremented by one.

Step 3: The trail I falsifies P, since $P_6|_I = 0$. The rule BP0F is applied. No model was found after taking the most recent relevant decision $\neg c$. It is flipped, and the model count associated with it is zero. The model count MC remains unaffected.

Step 4: A conflict in P is obtained, since $P_8|_I = 0$. The most recent relevant decision is a^d. It is flipped and the model count associated with it is the sum of the model counts associated with all flipped decisions which are unassigned, which is two. Since I is a counter-model of P, the model count MC remains unaffected.

Step 5: The trail I satisfies P. It is a total model of $\exists Y, S \,.\, P(X, Y, S)$. Rule BP1F is applied, and the most recent relevant decision c^d is flipped becoming $\neg c^{f(1)}$. The model count MC is incremented by one, since the found model is total.

Step 6: We have $N_6|_I = 0$, and $\pi(I, X)$ is a total model of $\exists Y, S \,.\, P(X, Y, S)$. Chronological backtracking occurs by executing rule BP1F. The most recent relevant decision literal is b^d. It is flipped and associated with the sum of the number of models found, which is one, and the model counts of the flipped decision literals which are unassigned during backtracking, which is only $\neg c^{f(1)}$. The resulting trail is $\neg a^{f(2)} \neg b^{f(2)}$. It contains two flipped decision literals each of which is associated with model count two. The number of models projected onto X represented by I is one, and the model count MC is incremented by one.

Step 7: A conflict in P has occurred, since $P_4|_I = 0$. Applying rule BP0F results in flipping the most relevant decision c^d, which is assigned with model count 0.

Step 8: The trail I falsifies P_2 and hence P. It contains no relevant decision literal indicating that the relevant search space has been explored exhaustively. The search terminates with $MC = 4$, which is correct. The models projected onto X are $a \neg b \neg c$, $a \neg b c$, $\neg a b c$, and $\neg a b \neg c$.

12.4 CONFLICT-DRIVEN CLAUSE LEARNING IN THE DUAL FORMULA

Our original idea was to analyze conflicts in N and to learn a clause allowing for pruning regions of the search space containing only models, similarly to CDCL, and to apply conflict-driven backjumping. However, it turns out that CDCL with conflict-driven backjumping implemented in state-of-the-art SAT solvers is applicable only in P: the search space is not traversed in an ordered manner, and while due to completeness of CDCL with non-chronological backtracking, the models of N will be found later in the search, this need not be the case for counter-models of N, which are the assignments we are interested in.

Suppose the trail contains a decision literal a^d and a conflict in N occurs. Further assume conflict analysis yields a unit clause (b). The model counter backtracks non-chronologically to decision level zero and propagates b. Propagating b at decision level zero prevents the check of all assignments containing $\neg b$ which were not yet tested and might result in missing conflicts in N. This comprises all right branches of the decision literals on the trail.

The following example clarifies this idea. Without loss of generation and for the sake of conciseness, we only consider $N(X, Y, T)$ where $T = Y = \emptyset$.

Example 12.3 (Missed models due to CDCL in the dual formula)**.** *Consider the propositional formula* $F(X, Y) = (\neg a \wedge b) \vee (\neg c \wedge \neg d) \vee (\neg c \wedge d)$ *with* $X = \{a, b, c, d\}$ *and* $Y = \emptyset$*. Its negation is* $N(X, Y, T) = N_1 \wedge N_2 \wedge N_3 = (a \vee \neg b) \wedge (c \vee d) \wedge (c \vee \neg d)$ *with* $T = \emptyset$*. Our dual model counter* Dualiza *reports the partial models* $\neg c$ *and* $\neg a b c$ *of* F*, which together represent* 10 *total models.*

After deciding $\neg a$ *using rule DX and deciding* b *by means of rule UNXY, the trail* $\neg a^d\, b^d$ *falsifies clause* N_1*, and the model* $m_1 = \neg a b$ *of* F *is found. Conflict analysis yields the conflict clause* $D_1 = (a \vee \neg b)$*, which is added to* N*. The assertion level of* D_1 *is one, and after backtracking to decision level one, the literal* $\neg b$ *is propagated with reason* D_1*. The trail is* $\neg a^d \neg b^{D_1}$*, and the residual of* N *under this trail is undefined. The decision* $\neg c^d$ *is taken using rule DX followed by deciding* $\neg d$ *by means of rule UNXY. The resulting trail* $\neg a^d \neg b^{D_1} \neg c^d \neg d^d$ *falsifies clause* N_2*. Only the most recent two decisions are involved in the conflict, and conflict analysis yields the conflict clause* $D_2 = (c \vee d)$*. One total model* $m_2 = \neg a \neg b \neg c \neg d$ *has been found. The assertion level of* D_2 *is two, and the resulting trail after backtracking and propagating* d *with reason* D_2 *is* $\neg a^d \neg b^{D_1} \neg c^d\, d^{D_2}$ *falsifying* N_3*, and* $m_3 = \neg a \neg b \neg c d$*. For conflict analysis, the conflicting clause* N_3 *is resolved with* D_2*, the reason of* d*, and the conflict clause* $D_3 = (c)$ *is learnt, which is unit and triggers backtracking to decision level zero and propagating* c *with reason* D_3*. The resulting trail is* c^{D_3}*. Clearly, the models* $a \neg c \neg d$ *and* $a \neg c d$ *of* F *can not be found, since* c *is propagated at decision level zero and will never be flipped.*

12.5 PROPAGATING INPUT LITERALS IN THE DUAL FORMULA

Let the current trail be I such that $N|_I$ contains a unit literal ℓ with $\mathsf{var}(\ell) \in X \cup Y$. Unlike in Abstract Dual #DPLL, the complement of ℓ is decided (rule UNXY). The immediate consequence is a conflict in N, upon which the model $I \neg\ell$ is recorded or counted and the decision $\neg\ell^d$ is flipped (rule BN0F or BN0L). Applying rules UNXY and BN0F (or BN0L) has the same effect as rule UPN in Abstract Dual #DPLL with the difference that in the latter the model $I \neg\ell$ is not computed explicitly but counted according to Equation 9.2, which saves one decision.

The motivation for designing rule UNXY in this manner was to use the same code for conflict analysis in both P and N. However, executing CDCL with conflict-driven backjumping after a conflict in N might result in a loss of models as shown in Section 12.4. It might therefore make sense to define propagation of input literals in N similarly to rule UN in Abstract Dual #DPLL but with projection support, which makes it challenging. Developing the relevant rules is out of the scope of this thesis and an interesting future research topic.

12.6 DUAL BLOCKING CLAUSES

In our framework, blocking clauses can not be used in dual mode, because they are added only to P. In fact, since the blocking clauses are not logically entailed by P and N remains unaltered, the duality property does not hold anymore. While the intuition was that prioritizing unit propagation in P over unit propagation in N would be sufficient to deal with this issue, our experiments have shown that this is not the case, and that the found models need be blocked in N, too, to avoid finding models multiple times.

Recall that a model of $N(X, Y, T)$ is a counter-model of $\exists S . P(X, Y, X)$, i. e., we want N to be satisfied whenever an assignment satisfying $\exists S . P(X, Y, X)$ is repeated. This is achieved by adding it with a logical disjunction to N: if m is a model of $\exists S . P(X, Y, S)$, we set $P' = P \wedge \neg m$ and $N' = N \vee m$. Notice that N' is not in CNF anymore. It can easily be verified that, if P and N comply with the duality property, $P \wedge \neg m$ and $N \vee m$ comply with the duality property as well. The idea is clarified by a simple example.

Example 12.4 (Dual blocking clauses). *Let the formulae $P(X, Y, S) = P_1 = (a \vee b)$ and $N(X, Y, T) = N_1 \wedge N_2 = (\neg a) \wedge (\neg b)$ be defined over $X = \{a, b\}, Y = S = T = \emptyset$. Obviously, P and N are the negation of each other. The assignment $m = a\,b$ is a total model of P. For blocking it, we set $P' = P \wedge (\neg a \vee \neg b)$ and $N' = N \vee (a \wedge b)$. If the assignment $a\,b$ is encountered again, the formula P is falsified and N evaluates to 1 under this assignment: the clause $\neg m$ is falsified, hence P' is falsified, too; the cube $(a \wedge b)$ evaluates to 1, hence N' evaluates to 1, too. Similarly, every assignment satisfying P' falsifies N': in fact, it satisfies the clause P_1 contained in P and falsifies the clauses N_1 and N_2 contained in N, since P and N comply with the duality property; it further satisfies $\neg m$, i. e., is either $a \neg b$, $\neg a\, b$, or $\neg a \neg b$, all of which falsify $(a \wedge b)$.*

We have extended our dual model counter and enumerator DualCountPro, which is presented in Chapter 13, with the new rules. In view of our implementation Dualiza, avoiding the combination of CNF and DNF was the main reason to try to refrain from adding models to N. A solution might be to transform N back into CNF after adding a model. The corresponding dual blocking clause encoding is developed in Section 20.6.

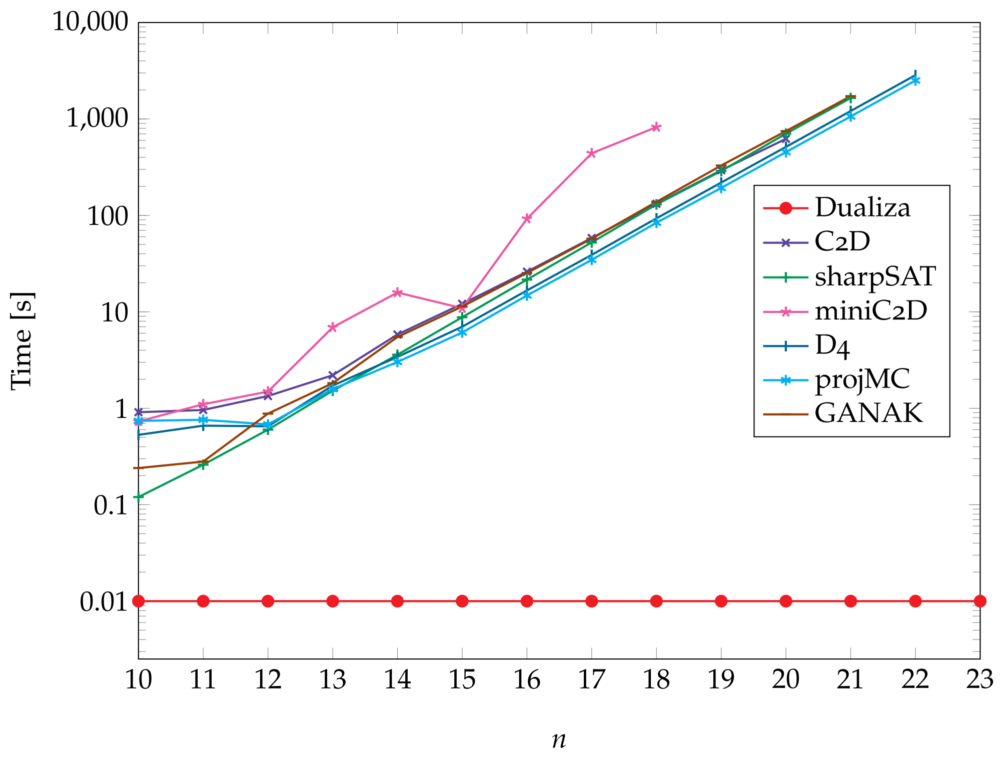


Figure 12.1: Solver comparison for non-decomposing formulae.

12.7 WHERE OUR DUAL APPROACH REALLY WINS

Our dual model counter DUALIZA outperforms sharpSAT [189] on a CNF formula which does not decompose into components after splitting on variables. Our example was the following, presented here for $n = 4$:

```
(x1 | x2 | x3 | x4) |
(x5 = x2 ^ x3 ^ x4) |
(x6 = x1 ^ x3 ^ x4) |
(x7 = x1 ^ x2 ^ x4) |
(x8 = x1 ^ x2 ^ x3)
```

There, "|" denotes disjunction ($\vee$) and "^" denotes XOR ($\oplus$). The number of models is $2^{2n} - 1$. The only falsifying assignment is the one assigning 0 to the first n variables and 1 to the second half.

Since the appearance of our paper, other #SAT solvers, namely projMCC [116] and GANAK [179], have been presented. Also, one goal in the paper was to compare DUALIZA with component-based model counters. We evaluate these newer solvers and repeat our experiments for DUALIZA and sharpSAT reported in the paper. For this thesis, we in addition report experiments with model counters implementing a knowledge compilation approach. The experiments were run on our cluster where each compute node has two Intel Xeon E5-2620 v4 CPUs running at 2.10 GHz with turbo-mode disabled. The time limit was set to 3600 seconds and the memory limit to 128 GB.

We run the experiments for $n \in \{10, \ldots, 26\}$ on Dualiza and the model counters C2D [55], sharpSAT [189], miniC2D [160], D4 [114], projMC [116], GANAK [179], listed in chronological order by their introduction. The results are visualized in

Table 12.2: Experimental results for nrp formulae. The abbreviation "SF" denotes a segmentation fault, and "TO" denotes a timeout.

n	dualiza	c2d	sharpsat	minic2d	d4	projmc	ganak
10	0.01	0.91	0.12	0.73	0.53	0.74	0.24
11	0.01	0.96	0.26	1.10	0.66	0.76	0.28
12	0.01	1.34	0.60	1.49	0.65	0.68	0.88
13	0.01	2.20	1.52	6.91	1.72	1.58	1.82
14	0.01	5.82	3.59	15.86	3.41	3.02	5.48
15	0.01	12.10	8.79	10.91	7.01	6.11	11.35
16	0.01	26.08	21.58	92.20	16.67	14.77	25.28
17	0.01	58.30	52.47	440.55	38.96	34.64	57.40
18	0.01	131.89	129.90	823.51	93.20	83.91	138.15
19	0.01	294.01	288.51	TO	219.01	192.72	329.20
20	0.01	622.36	704.11	TO	513.19	453.24	748.71
21	0.01	SF	1648.97	TO	1209.26	1062.46	1724.12
22	0.01	MO	TO	TO	2846.74	2507.87	TO
23	0.01	MO	TO	TO	TO	SF	TO
24	0.01	MO	TO	TO	TO	SF	TO
25	0.01	MO	TO	TO	TO	TO	TO
26	0.01	MO	TO	TO	TO	TO	TO
100	0.10	-	-	-	-	-	-
1000	9.22	-	-	-	-	-	-
2000	37.42	-	-	-	-	-	-
3000	84.28	-	-	-	-	-	-
4000	149.05	-	-	-	-	-	-
5000	233.66	-	-	-	-	-	-

Figure 12.1. On the x axis n is given, on the y axis the solving time in seconds. Dualiza counts the models for $n \in \{10, \ldots, 26\}$ in 0.01 seconds. The other solvers show an exponential behavior. The solvers D4 and projMC perform best and compute the model count for n up to 22 within one hour. sharpSAT and GANAK manage to count the models for n up to 21, while C2D counts the models for n up to 20 and miniC2D for n up to 18. The experiments were repeated for $n \in \{100, 1000, 2000, 3000, 4000, 5000\}$ on our solver Dualiza. In this run, Dualiza was able to count the models for $n = 5000$ in 233.66 seconds. Detailed results are presented in Table 12.2. The abbreviation "SF" denotes a segmentation fault, "TO" denotes a timeout, and "MO" means that the solver run out of memory.

All solvers except Dualiza show exponential behavior. The solver C2D reports a segmentation fault (SF) for $n = 21$ and "out of memory for storing NNF" ("MO") for $n \in \{22, \ldots, 26\}$. Except for $n \in \{10, 15\}$, C2D is faster than miniC2D, and with growing n this lead increases. The reason might be twofold. On the one hand C2D compiles the formula in question into a d-DNNF, while miniC2D compiles it into an SDD [160]; on the other hand nrp formulae do not partition into disconnected components after splitting on variables. Notice that C2D ran out of memory after 836.00 to 862.92 seconds.[1]

sharpSAT counts the models for n up to 21 and is slightly faster than GANAK. This is interesting, since Sharma, Roy, Soos, and Meel [179] in their evaluation found that GANAK outperformed sharpSAT significantly on some instances. One possible reason for this discrepancy might be given by the structure of the formulae. There is evidence that real-world benchmark instances partition into disconnected components after splitting on variables, while nrp formulae don't.

The solvers projMC and D4 exhibit a similar behavior and succeed for n up to 22. While D4 runs out of time for $n \in \{23, \dots, 26\}$, projMC produces a segmentation fault for $n \in \{23, 24\}$ and runs out of time for $n \in \{25, 26\}$. We ran experiments for $n \in \{100, 1000, 2000, 3000, 4000, 5000\}$ only on DUALIZA. The results are shown in the lower part of Table 12.2. DUALIZA computes the model count for $n = 5000$ in 233.66 seconds, qualitatively confirming our previous result presented in Section 11.6.

1 These times are not reported in the table.

13 DUALCOUNTPRO – A DUAL MODEL COUNTER IN PROLOG

To check the rules of our dual model counting frameworks (Chapter 9 and Chapter 11), we implemented DualCountPro, a proof of concept in SWI-Prolog [199] making use of the PIE environment [198]. Our goal was to perform a one-to-one check of our rules, and SWI-Prolog provides an elegant solution. We briefly touch upon the main architecture of our tool DualCountPro, a parametric model counter and model enumerator that also supports projection, before presenting the implementation of rule BN0L in more detail.

In an early stage of our work on dual model counting, we developed a classification for organizing different variants of our framework based on the following binary features:

D_IP. If set to 1, irrelevant and primal variables are treated with the same priority with respect to decisions. If set to 0, primal variables are decided only after all irrelevant input variables are assigned.

D_T. If set to 1, deciding dual variables is allowed.

J_P. Turns conflict-driven backjumping in P on if set to 1 and off otherwise.

J_N. Enables or disables conflict-driven backjumping in N.

L. Controls whether blocking clauses are added to P or not.

F. If set to 1, flipping and discounting is enabled.

Check_M. Turns checking the models in M on or off. If set to 1, this feature has the same effect as adding the found models to N and corresponds to using dual blocking clauses if Check_R is set to 1 as well.

Check_R. Checks whether the refutation is already contained in R. This option corresponds to adding blocking clauses to P.

A framework is characterized by a tuple (D_IP, D_T, J_P, J_N, L, F, Check_M, Check_R), where each element can be either 1, 0, or irrelevant, denoted by x. Rule BN0L, depicted in Figure 13.2, could therefore be represented as $(1, 0, 1, 0, x, x, 0, 1)$.

The options can be combined with all features. They allow for a more fine-grained characterization of a framework. The following options are defined:

Dual. If set to 1, DualCountPro runs in dual mode. This option is very useful in comparing execution traces in dual and non-dual mode.

U_N. If set to 1, unit propagation is allowed in N.

Top_rules. This option was created for disabling the use of the 1 rules. The idea was to check whether it was was possible to solve a formula only by conflicts.

Enumerate. If set to 1, the models are enumerated.

Count. If set to 1, the total models are counted. Enumerate and Count can also be used simultaneously.

The features defined for our classification gave rise to some of the parameters DualCountPro takes as input, and this versatile architecture turned out to be very useful for playing around with different variations of our framework.

Formulae are represented as lists of clauses, and clauses and cubes are represented as lists of literals. DualCountPro maintains lists for the primal formula (denoted by P) and the dual formula (N) as well as the detected models (M) and their refutations (R), which are their negations. In addition, the corresponding residuals under the current trail are stored in separate lists (RP, RN, RM, and RR).

Sets of variables are also represented as lists. DualCountPro keeps lists for the relevant input variables (X_vars) and the irrelevant input variables (Y_vars) as well as for the primal variables (S_vars) and dual variables (T_vars), which are the variables introduced by the transformation into CNF of the input formula F into P and its negation $\neg F$ into N. An additional list is defined containing the variables in $Y \cup S$ (YS_vars). It is used in place of Y_vars and S_vars if the irrelevant inputs and primal variables are decided with the same priority, i. e., D_IP = 1. No decision heuristic is implemented but we permute the variable lists randomly in order to be to check our rules with different variable orderings, as we choose the decision variables according to their ordering on the trail. The trail is represented as a list of annotated literals (not shown).

States are represented as state(I, M, R, MC). They contain only elements which are altered by the predicate apply_rule. Neither P nor N occur, because they are not altered but M and R are taken into account additionally. We suffix the elements of input states, which are states before executing a rule, with "_in" and the ones of output states, i. e., states after rule execution, with "_out".

DualCountPro executes the loop count shown in Figure 13.1. First it computes RP, RN, RM, and RR, if needed (lines 1–14). Then it checks the rules sequentially for their applicability until it finds an appropriate rule and succeeds with its application (lines 15–22). A rule consists of a predicate apply_rule with the framework features and options specified accordingly. If after executing a rule the input state differs from the output state, the output state becomes the input state for the next iteration (lines 23–34). Otherwise, the computation terminates (lines 35–38).

The implementations of our rules depicted in Section 11.4 do not differ significantly from each other, and we present rule BN0L shown in Figure 13.2. Its implementation is shown in Figure 13.3. The predicate apply_rule represents one step from our calculus. It takes as inputs the framework features and options, the formulae P and N, the various variable lists and residuals just computed as well as the current input state (lines 1–5). Its output is the state altered by its execution (line 6). The trail I_in and the number of models found so far MC_in correspond to the trail $I\,\ell^d\,I'$ and the DNF formula M in rule BN0L.

Rule BN0L is applicable if $N|_{I\,\ell^d\,I'}$ contains the empty clause, i. e., if RN contains the empty clause, which is expressed as a membership test with the empty list of literals [] representing the empty clause (line 9). DualCountPro does not add blocking clauses to P but records them in R. Accordingly it checks whether RR contains the empty clause (line 11). If this is the case, the check fails, and apply_rule fails. If RR does not contain the empty clause, the most recent decision with literal in X_vars is flipped (line 14), which corresponds to flipping the decision ℓ^d in rule BN0L. If I_in contains no relevant decision, the predicate flip_most_recent_decision fails, and hence apply_rule fails too. Otherwise, the literals are retrieved from I_in (line 17 and projected onto the relevant variables X_vars and the model count updated (lines 22–24). Num_r_vars denotes

```
count(
   framework(D_IP, D_T, J_P, J_N, L, F, Check_M, Check_R),
   options(Dual, U_N, Top_rules, Enumerate, Count),
   P, N, R_vars, I_vars, P_vars, D_vars, IP_vars,
   state(I_in, M_in, R_in, MC_in),
   Run, M_final, R_final, MC_final
) :-
    % compute reducts if needed, otherwise assign them []
    reduct_cnf(P, I_in, RP),
    (    Dual == 1 ->
         reduct_cnf(N, I_in, RN)
    ;    RN = []
    ),
    (    Check_M == 1 ->
         reduct_dnf(M_in, I_in, RM)
     ;    RM = []
    ),
    (    Check_R == 1 ->
         reduct_cnf(R_in, I_in, RR)
    ;    RR = []
    ),
    apply_rule(
       framework(D_IP, D_T, J_P, J_N, L, F, Check_M, Check_R),
       options(Dual, U_N, Top_rules, Enumerate, Count),
       P, N, R_vars, I_vars, P_vars, D_vars, IP_vars,
       RP, RN, RM, RR,
       state(I_in, M_in, R_in, MC_in),
       state(I_out, M_out, R_out, MC_out)
   ),
   (    (    not(I_out == I_in)
        ;    I_out == I_in,
             flip_last_spec_decision(I_out, I_tmp, R_vars, Dec_var)
        ),
        Run_next is Run + 1,
        count(
            framework(D_IP, D_T, J_P, J_N, L, F, Check_M, Check_R),
            options(Dual, U_N, Top_rules, Enumerate, Count),
            P, N, R_vars, I_vars, P_vars, D_vars, IP_vars,
            state(I_out, M_out, R_out, MC_out),
            Run_next, M_final, R_final, MC_final
        )
        M_final = M_out,
        R_final = R_out,
        MC_final is MC_out,
        Run_next is Run,
   ).
```

Figure 13.1: SWI-Prolog code for outer loop of DualCountPro.

BN0L: $(P, N, I\ell^d I', M) \rightsquigarrow_{\mathsf{BN0L}} (P \wedge D, N, I\bar{\ell}, M + m'')$ if $\emptyset \in N|_{I\ell I'}$ and
$\mathsf{var}(\ell) \in X$ and $\mathsf{var}(\mathsf{decs}(I')) \cap X = \emptyset$ and $m'' = 2^{|X - I\ell I'|}$ and
$D = \pi(\neg \mathsf{decs}(I\ell), X)$

Figure 13.2: Rule BN0L from the dual model counting calculus.

```
% BN0L - backtracking upon conflict in N with learning
%
% Is applicable if
% - RN contains the empty clause and
% - RR does not contain the empty clause
% - I_in contains a decision literal with variable in X_vars
%
apply_rule(
    framework(D_IP, D_T, J_P, J_N, L, F, Check_M, Check_R),
    options(Dual, U_N, Top_rules, Enumerate, Count),
    P, N, X_vars, Y_vars, S_vars, T_vars,
    RP, RN, RM, RR,
    state(I_in, M_in, R_in, MC_in),
    state(I_out, M_out, R_out, MC_out)
) :-
    % conflict in N
    member([], RN),
    % model was not yet found
     not(member([], RR))
    % flip the most recent decision with variable in X_vars
    % fails if no relevant decision is left on I_in
    flip_most_recent_decision(I_in, I_out, X_vars, Dec_var),
    % most relevant decision was flipped
    % compute model projected onto the relevant variables
    assigned_lits(I_in, AL),
    project(AL, X_vars, Model),
    % update list of models
    add_first(Model, M_in, M_out),
    % update model count
    length(Model, Model_length),
    length(X_vars, Num_r_vars),
    MC_out is MC_in + 2^(Num_r_vars - Model_length),
    % update refutations
    negate_cube(Model, Refutation),
    add_first(Refutation, R_in, R_out).
```

Figure 13.3: SWI-Prolog code for rule BN0L.

the number of relevant variables and is defined as a global variable and computed during initialization. Finally, the corresponding blocking clause is computed and added to the list of refutations (lines 26–27).

Part IV

CHRONOLOGICAL CONFLICT-DRIVEN CLAUSE LEARNING FOR PROPOSITIONAL MODEL COUNTING

14

PAPER 3: BACKING BACKTRACKING

PUBLISHED. In: *Theory and Applications of Satisfiability Testing – SAT 2019 – 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9–12, 2019, Proceedings*. Ed. by Mikolás Janota and Inês Lynce. Vol. 11628. Lecture Notes in Computer Science. Springer, 2019, pp. 250–266. DOI: 10.1007/978-3-030-24258-9_18.

AUTHORS. Sibylle Möhle and Armin Biere.

ACKNOWLEDGMENTS. Supported by the Austrian Science Fund (FWF) grant S11408-N23 (RiSE) and by the LIT Secure and Correct Systems Lab funded by the State of Upper Austria.

ABSTRACT. Non-chronological backtracking was considered an important and necessary feature of conflict-driven clause learning (CDCL). However, a SAT solver combining CDCL with chronological backtracking succeeded in the main track of the SAT Competition 2018. In that solver, multiple invariants considered crucial for CDCL were violated. In particular, decision levels of literals on the trail were not necessarily increasing anymore. The corresponding paper presented at SAT 2018 described the algorithm and provided empirical evidence of its correctness, but a formalization and proofs were missing. Our contribution is to fill this gap. We further generalize the approach, discuss implementation details, and empirically confirm its effectiveness in an independent implementation.

14.1 INTRODUCTION

Most state-of-the-art SAT solvers are based on the CDCL framework [125, 127]. The performance gain of SAT solvers achieved in the last two decades is to some extent attributed to combining conflict-driven backjumping and learning. It enables the solver to escape regions of the search space with no solution.

Non-chronological backtracking during learning enforces the lowest decision level at which the learned clause becomes unit and then is used as a reason. While backtracking to a higher level still enables propagation of a literal in the learned clause, the resulting propagations might conflict with previous assignments. Resolving these conflicts introduces additional work which is prevented by backtracking non-chronologically to the lowest level [157].

However, in some cases a significant amount of the assignments undone is repeated later in the search [149, 188], and a need for methods to save redundant work has been identified. Chronological backtracking avoids redundant work by keeping assignments which otherwise would be repeated at a later stage of the

Chronological and non-chronological CDCL:

Trail:	The assignment trail contains neither complementary pairs of literals nor duplicates.
ConflictLower:	The assignment trail preceding the current decision level does not falsify the formula.

Non-chronological CDCL only:

Propagation:	On every decision level preceding the current decision level all unit clauses are propagated until completion.
LevelOrder:	The literals are ordered on the assignment trail in ascending order with respect to their decision level.
ConflictingClause:	At decision levels greater than zero the conflicting clause contains at least two literals with the current decision level.

Figure 14.1: The CDCL invariants listed in the box are usually considered crucial to CDCL. By combining CDCL with chronological backtracking, the last three are violated.

search. As our experiments show, satisfiable instances benefit most from chronological backtracking. Thus this technique should probably also be seen as a method to optimize SAT solving for satisfiable instances similar to [9, 156].

The combination of chronological backtracking with CDCL is challenging since invariants classically considered crucial to CDCL cease to hold. Nonetheless, taking appropriate measures preserves the solver's correctness, and the combination of chronological backtracking and CDCL appeared to be a winning strategy: The SAT solver Maple_LCM_Dist_ChronoBT [150] was ranked first in the main track of the SAT Competition 2018.

In Figure 14.1 we give invariants classically considered crucial to CDCL which are relevant for the further discussion. Our aim is to demonstrate that although some of them do not hold in [149], the solving procedure remains correct.

Clearly, if upon conflict analysis the solver jumps to a decision level higher than the asserting level, invariant Propagation is violated. Measures to fix potential conflicting assignments were proposed in [149] which in addition violated invariants LevelOrder and ConflictingClause. The algorithm's correctness as well as its efficiency were empirically demonstrated. However, a formal treatment with proofs was not provided.

Our contribution. Our main contribution consists in providing a generalization of the method presented in [149] together with a formalization. We prove that despite violating some of the invariants given above, the approach is correct. Our experiments confirm the effectiveness of chronological backtracking with an independent implementation in our SAT solver CaDiCaL [19].

14.2 preliminaries

Let F be a formula over a set of variables V. A *literal* ℓ is either a variable $v \in V$ or its negation $\neg v$. The variable of ℓ is obtained by $V(\ell)$. We denote by $\overline{\ell}$ the *complement* ℓ, i.e., $\overline{\ell} = \neg\ell$, and assume $\neg\neg\ell = \ell$. We consider formulae in conjunctive normal form (CNF) defined as conjunctions of clauses which are disjunctions of literals. We write $C \in F$ if C is a clause in F and $\ell \in C$ for a literal ℓ occurring in C interpreting F as a set of clauses and C as a set of literals. We use set notation for formulae and clauses where convenient.

We call *trail* a sequence of literals with no duplicated variables and write $I = \ell_1 \dots \ell_n$. We refer to an element ℓ of I by writing $\ell \in I$ interpreting I as a set of literals and denote the set of its variables by $V(I)$. Trails can be concatenated, $I = JK$, assuming $V(J) \cap V(K) = \emptyset$. We denote by $\tau(I, \ell)$ the position of the literal ℓ on the trail I. A *total assignment* is a mapping from V to the truth values 1 and 0. A trail may be interpreted as a *partial assignment* where $I(\ell) = 1$ iff $\ell \in I$. Similarly, $I(C)$ and $I(F)$ are defined.

The *residual* of the formula F under the trail I, denoted by $F|_I$, is obtained by replacing in F the literals ℓ where $V(\ell) \in I$ with their truth value. We define the residual of a clause in an analogous manner. The empty clause and the literal assigned truth value 0 are denoted by $\bot$, the empty formula by $\top$. If $I(F) = \top$, i.e., $F|_I = \top$, we say that I *satisfies* F and call I a *model* of F. If $I(C) = \bot$ for a clause $C \in F$, i.e., $C|_I = \bot$ and hence $F|_I = \bot$, we say that I *falsifies* C (and therefore F) and call C the *conflicting clause*.

We call *unit clause* a clause $\{\ell\}$ containing one single literal ℓ which we refer to as *unit literal*. We denote by $\mathsf{units}(F)$ the sequence of unit literals in F and extend this notion to the residual of F under I by writing $\mathsf{units}(F|_I)$. We write $\ell \in \mathsf{units}(F|_I)$ for referring to the unit literal ℓ in the residual of F under I.

14.3 generalizing cdcl with chronological backtracking

In classical CDCL SAT solvers based on *non-chronological* backtracking [125] the trail reflects the order in which literals are assigned. The trail is used during conflict analysis to simplify traversal of the implication graph in reverse assignment order and in general during backtracking to undo assignments in the proper reverse assignment order.

In CDCL with non-chronological backtracking, the trail is partitioned into subsequences of literals between decisions in which all literals have the same decision level. Each subsequence starts with a decision literal and extends until the last literal before the next decision. Literals assigned before any decision may form an additional subsequence at decision level zero.

After adding *chronological* backtracking to CDCL as described in [149], the trail is not partitioned in the same way but subsequences of the same decision level are interleaved, while still respecting the assignment order.

Let $\delta\colon V \mapsto \mathbb{N} \cup \{\infty\}$ return the decision level of a variable v in the set of variables V, with $\delta(v) = \infty$ if v is unassigned. This function is updated whenever a variable is either assigned or unassigned. The function δ is extended to literals ℓ, clauses C and trails I by defining $\delta(\ell) = \delta(V(\ell))$, $\delta(C) = \mathsf{max}\{\delta(\ell) \mid \ell \in C\}$ for $C \neq \bot$, and $\delta(I) = \mathsf{max}\{\delta(\ell) \mid \ell \in I\}$. We further define $\delta(\bot) = 0$.

Given a set of literals L, we denote by $\delta(L) = \{\delta(\ell) \mid \ell \in L\}$ the set containing the decision levels of its elements. The function δ updated with decision level d assigned to $V(\ell)$ is denoted by $\delta[\ell \mapsto d]$. Similarly, $\delta[I \mapsto \infty]$ represents the function δ where all literals on the trail I are unassigned. In the same manner, $\delta[V \mapsto \infty]$ assigns all variables in V to decision level ∞. We may write $\delta \equiv \infty$ as a shortcut. The function δ is left-associative. We write $\delta[L \mapsto \infty][\ell \mapsto b]$ to express that the function δ is updated by first unassigning all literals in a sequence of literals L and then assigning literal ℓ to decision level b.

For the sake of readability, we write $J \leqslant I$ where J is a subsequence of I and the elements in J have the same order as in I and $J < I$ when furthermore $J \neq I$. We denote by $I_{\leqslant b}$ the subsequence of I containing exactly the literals ℓ where $\delta(\ell) \leqslant b$.

Due to the interleaved trail structure we need to define decision literals differently than in CDCL. We refer to the set consisting of all decision literals on I by writing $\mathsf{decs}(I)$ and define a *decision literal* ℓ as

$$\ell \in \mathsf{decs}(I) \quad \text{iff} \quad \ell \in I,\ \delta(\ell) > 0,\ \forall k \in I \,.\, \tau(I,k) < \tau(I,\ell) \Rightarrow \delta(k) < \delta(\ell) \qquad (14.1)$$

Thus, the decision level of a decision literal $\ell \in I$ is strictly higher than the decision level of any literal preceding it on I. If $C|_I = \{\ell\}$ for a literal ℓ, then ℓ is not a decision literal. The set $\mathsf{decs}(I)$ can be restricted to decision literals with decision level lower or equal to i by writing $\mathsf{decs}_{\leqslant i}(I) = \mathsf{decs}(I_{\leqslant i})$.

As in [155] we use an abstract representation of the assignment trail I by writing $I = I_0 \ell_1 I_1 \dots \ell_n I_n$ where $\{\ell_1, \dots, \ell_n\} = \mathsf{decs}(I)$. We denote by $\mathsf{slice}(I,i)$ the i-th *slice* of I, i.e., the subsequence of I containing all literals ℓ with the same decision level $\delta(\ell) = i$. The i-th *block*, denoted by $\mathsf{block}(I,i)$, is defined as the subsequence of I starting with the decision literal with decision level i and extending until the last literal before the next decision:

$$\begin{aligned} \mathsf{slice}(I,i) &= I_{=i} \\ \mathsf{block}(I,i) &= \ell_i I_i \end{aligned}$$

Note that in general $I_{=i} \neq I_i$, since I_i (due to the interleaved structure of the trail I) may contain literals with different decision levels, while this is not the case in $I_{=i}$. In particular, there might be literals with a lower decision level than some literal preceding them on the trail. We call these literals *out-of-order literals*. Contrarily to classical CDCL with non-chronological backtracking, upon *backtracking* to a decision level b, blocks must not be discarded as a whole, but only the literals in $\mathsf{slice}(I,i)$ where $i > b$ need to be unassigned.

Consider the trail I on the left hand side of Figure 14.2 over variables $\{1, \dots, 5\}$ (in DIMACS format) where τ represents the position of a literal on I and δ represents its decision level:

Literals 1 and 3 were propagated at decision level zero, literal 5 was propagated at decision level one. The literals 3 and 5 are out-of-order literals: We have $\delta(2) = 1 > 0 = \delta(3)$, whereas $\tau(I,2) = 1 < 2 = \tau(I,3)$. In a similar manner, $\delta(4) = 2 > 1 = \delta(5)$, and $\tau(I,4) = 3 < 4 = \tau(I,5)$. Moreover, $I_{\leqslant 1} = 1\,2\,3\,5$, $\mathsf{decs}(I) = 2\,4$, $\mathsf{decs}_{\leqslant 1}(I) = 2$, $\mathsf{slice}(I,1) = 2\,5$, and $\mathsf{block}(I,1) = 2\,3$.

Upon backtracking to decision level one, the literals in $\mathsf{slice}(I,2)$ are unassigned. The resulting trail is visualized in the middle of Figure 14.2. Note that since the assignment order is preserved, the trail still contains one out-of-order literal, namely 3. Backtracking to decision level zero unassigns all literals in $\mathsf{slice}(I,2)$ and $\mathsf{slice}(I,1)$ resulting in the trail in which all literals are placed in order depicted on the right hand side.

τ	0	1	**2**	3	**4**
I	1	2	**3**	4	**5**
δ	0	1	**0**	2	**1**

τ	0	1	**2**	3
I	1	2	**3**	5
δ	0	1	**0**	1

τ	0	1
I	1	3
δ	0	0

Figure 14.2: In the trail I on the left, from the three trails shown, literals 3 and 5 are placed out of order. In fact, their decision level δ is lower than the decision level of a literal preceding them on the trail, i.e., with lower position τ. The trails in the middle and on the right hand side show the results of backtracking to decision levels 1 and 0. When backtracking to the *backtrack level* b, only literals ℓ with $\delta(\ell) > b$ are removed from the trail, while the assignment order is preserved.

True: $(F, I, \delta) \leadsto_{\text{True}} \text{SAT}$ if $F|_I = \top$

False: $(F, I, \delta) \leadsto_{\text{False}} \text{UNSAT}$ if exists $C \in F$ with $C|_I = \bot$ and $\delta(C) = 0$

Unit: $(F, I, \delta) \leadsto_{\text{Unit}} (F, I\ell, \delta[\ell \mapsto a])$ if $F|_I \neq \top$ and $\bot \notin F|_I$ and exists $C \in F$ with $\{\ell\} = C|_I$ and $a = \delta(C \setminus \{\ell\})$

Jump: $(F, I, \delta) \leadsto_{\text{Jump}} (F \wedge D, PK\ell, \delta[L \mapsto \infty][\ell \mapsto j])$ if exists $C \in F$ with $PQ = I$ and $C|_I = \bot$ such that $c = \delta(C) = \delta(D) > 0$ and $\ell \in D$ and $\ell|_Q = \bot$ and $F \models D$ and $j = \delta(D \setminus \{\ell\})$ and $b = \delta(P)$ and $j \leqslant b < c$ and $K = Q_{\leqslant b}$ and $L = Q_{>b}$

Decide: $(F, I, \delta) \leadsto_{\text{Decide}} (F, I\ell, \delta[\ell \mapsto d])$ if $F|_I \neq \top$ and $\bot \notin F|_I$ and $\text{units}(F|_I) = \emptyset$ and $V(\ell) \in V$ and $\delta(\ell) = \infty$ and $d = \delta(I) + 1$

Figure 14.3: In the transition system of our framework non-terminal states (F, I, δ) consist of a CNF formula F, the current trail I and the decision level function δ. The rules formalize termination (True and False), backtracking (Jump), unit propagation (Unit) and picking decisions (Decide).

14.4 CALCULUS

We devise our calculus as a transition system over a set of states S, a transition relation $\leadsto \subseteq S \times S$ and an initial state s_0. Non-terminal states are described by (F, I, δ) where F denotes a formula over variables V, I denotes the current trail and δ refers to the decision level function.

The *initial* state is given by $s_0 = (F, \varepsilon, \delta_0)$. In this context, F is the original formula, ε denotes the empty trail and $\delta_0 \equiv \infty$. The *terminal* state is either SAT or UNSAT expressing satisfiability or unsatisfiability of F. The transition relation $\leadsto$ is defined as the union of transition relations $\leadsto_{\text{R}}$ where R is either True, False, Unit, Jump or Decide. These rules are listed in Figure 14.3. We first explain the intuition behind these rules before proving correctness in Section 14.5:

True / False. If $F|_I = \top$, F is satisfiable and the search terminates in the state SAT (rule True). If $F|_I = \bot$, a clause $C \in F$ exists where $I(C) = \bot$. The *conflict level* is $\delta(C) = 0$. Obviously $I_{\leqslant 0}(F) = \bot$ and consequently F is unsatisfiable. Then the procedure terminates in state UNSAT (False).

Unit. Propagated unit literals are assigned the maximum decision level of their reason which may be lower than the current decision level. Requiring that the residual of F under I is conflict-free ensures that invariant ConflictLower holds.

Jump. We have $F|_I = \bot$, i.e., there exists a clause $C \in F$ for which we have $I(C) = \bot$. Since the conflict level is $\delta(C) = c > 0$, there is a decision left on I. We assume to obtain a clause D implied by F (usually through conflict analysis) with $\delta(D) = c > 0$ whose residual is unit, e.g., $\{\ell\}$, at *jump level* $j = \delta(D \setminus \{\ell\})$, the second highest decision level in D. In fact, the residual of D under the trail is unit at any backtrack level b where $j \leqslant b < c \leqslant d$, with $d = \delta(I)$ denoting the current decision level. Using D as a reason, we may backtrack to any of these decision levels. Remember that the decision levels on the trail do not have to be sorted in ascending order and that upon backtracking only the literals in the i-th slice with $i > b$ need to be unassigned as discussed in Section 14.3. After backtracking we propagate ℓ and assign it decision level j to obtain $\delta(PK\ell) = b$.

Note. If the conflicting clause C contains exactly one literal ℓ assigned at conflict level c, its residual is unit at decision level $c - 1$. The solver therefore could backtrack to decision level $c - 1$ and propagate ℓ. An optimization is to use $D = C$ as reason saving the computational effort of conflict analysis. This corresponds to learning C instead of a new clause D and is a special case of rule Jump. It is also explicitly included in the pseudocode in [149].

Decide. If F is neither satisfied nor falsified and there are no unit literals in $F|_I$, an unassigned variable is assigned. Invariants ConflictLower and Propagation hold.

Example. As pointed out above, the conflicting clause C may contain one single literal ℓ assigned at decision level c. While according to the pseudocode in [149] backtracking is executed to the second highest decision level j in C, the implementation MAPLE_LCM_DIST_CHRONOBT backtracks chronologically to decision level $c - 1$, which may be higher than j. We adopt this strategy in our own solver, but unlike MAPLE_LCM_DIST_CHRONOBT we eagerly propagate ℓ and assign it decision level j, as described in the explanation of rule Jump above.

The authors of [149] focused on unit propagation and backtracking, and an in-depth discussion of the case in which the conflicting clause contains exactly one literal at conflict level is missing. We fill this gap and explain our calculus in detail by means of an example for this case. We generated this example with our model-based tester Mobical for CADICAL based on ideas in [8, 154]. Our example is larger and provides a good intuition regarding the occurrence of multiple nested decision levels on the trail as well as its effect on backtracking.

We represent variables by natural numbers and consider a formula F over variables $\{1, \ldots, 48\}$ as above where negative numbers encode negated variables. We further use set notation for representing clauses. Consider the following assignment trail excerpt where the trail I is represented as a sequence of literals, τ denotes the position of a literal on I and δ its decision level:

τ	$\cdots$	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
I	$\cdots$	4	5	30	47	15	18	6	−7	−8	45	9	38	−23	17	44	−16
δ	$\cdots$	3	4	4	4	4	4	5	5	5	5	6	6	6	6	6	6
C		{			−47,										**−17**,	**−44**	}
D		{		−30,	−47,		−18,							**23**			}

Initially, the literals are placed in order on I, i.e., they are sorted in ascending order with respect to their decision level. At this point, a conflict occurs. The conflicting clause is $C = \{-47, -17, -44\}$ depicted below the trail containing two literals at conflict level $c = \delta(C) = 6$, i.e., -17 and -44 depicted in boldface in the above outline. Conflict analysis in our implementation learned the clause $D = \{-30, -47, -18, 23\}$ where $\delta(-30) = \delta(-47) = \delta(-18) = 4$ and $\delta(23) = 6$. Since $\delta(D) = c = 6$ and $j = \delta(D \setminus \{23\}) = 4$, the solver in principle would be free to backtrack to either decision level 4 or 5.

Let the solver backtrack *chronologically* to decision level 5 where D becomes unit, specifically $\{23\}$. The position on the trail the solver backtracks to is marked with a vertical dotted line. Accordingly all literals with decision level higher than 5 are removed from I (literals at positions higher than 13). Then literal 23 is propagated. The jump level is $j = 4$, hence literal 23 is assigned decision level 4 out of order. Literal -38 is propagated due to $\mathsf{reason}(-38) = \{-15, -23, -38\}$ (not shown). Since $\delta(-15) = \delta(-23) = 4$, literal -38 is assigned decision level 4. Then literal -9 is propagated with $\mathsf{reason}(-9) = \{-45, 38, -9\}$ with $\delta(-45) = 5$ and $\delta(38) = 4$. Thus, -9 is assigned decision level 5. The resulting trail is

τ	$\cdots$	4	5	6	7	8	9	10	11	12	13	**14**	**15**	16
I	$\cdots$	4	5	30	47	15	18	6	−7	−8	45	**23**	**−38**	−9
δ	$\cdots$	3	4	4	4	4	4	5	5	5	5	**4**	**4**	5

where the literals 23 and -38 (depicted in boldface) are placed out of order on I. Later in the search we might have the following situation:

τ	$\cdots$	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
I	$\cdots$	18	6	−7	−8	45	23	−38	−9	10	−11	13	16	−17	−25	42	12	−41
δ	$\cdots$	4	5	5	5	5	4	4	5	6	7	5	4	4	4	4	5	5
C		{												17,		−42,	**−12**	}

The first assignment after analyzing the last conflict is placed right after the dashed vertical line. Again, a conflict occurs. Let $C = \{17, -42, -12\}$ be the conflicting clause. The conflict level is $\delta(C) = 5$ and the decision level of I is $\delta(I) = 7$. Clause C contains exactly one literal at conflict level, namely -12 depicted in boldface. The solver backtracks to decision level $c - 1 = 4$ marked with a thick solid line. After removing from I all literals with decision level higher than 4 and propagating literal -12, the resulting trail is

τ	$\cdots$	9	10	11	12	13	14	15	16
I	$\cdots$	18	23	−38	16	−17	−25	42	−12
δ	$\cdots$	4	4	4	4	4	4	4	4

Note that as discussed above we use $D = C \in F$ without actually adding it.

14.5 PROOFS

For proving the correctness of our method, we first show that the system terminates in the correct state which can be done in a straightforward manner. Proving that the system always makes progress is more involved. Last we prove that our procedure terminates by showing that no infinite sequence of states is generated. By $\delta(\mathsf{decs}(I))$ we denote the set consisting of the decision levels of the set of decision literals on I. We start by proving the following invariants:

(1) $\forall k, \ell \in \mathsf{decs}(I) \,.\, \tau(I,k) < \tau(I,\ell) \implies \delta(k) < \delta(\ell)$

(2) $\delta(\mathsf{decs}(I)) = \{1, \dots, \delta(I)\}$

(3) $\forall n \in \mathbb{N} \,.\, F \wedge \mathsf{decs}_{\leqslant n}(I) \models I_{\leqslant n}$.

Lemma 14.1 (Invariants). *Invariants* (1) – (3) *hold in non-terminal states.*

Proof. The proof is carried out by induction over the number of rule applications. We assume Invariants (1) – (3) hold in a non-terminal state (F, I, δ) and show that they are still met after the transition to another non-terminal state for all rules.

Unit: The trail I is extended by a literal ℓ. We need to show that ℓ is not a decision literal. To this end it is sufficient to consider the case where $a > 0$. There exists a clause $C \in F$ with $\{\ell\} = C|_I$. Since $a = \delta(C \setminus \{\ell\})$, there exists a literal $k \in C$ where $k \neq \ell$ and such that $\delta(k) = a$. Obviously, k was assigned prior to ℓ and $\tau(I,k) < \tau(I,\ell)$. Since $\delta(k) = \delta(\ell)$ and by the definition of decision literal in Equation 14.1, ℓ is not a decision literal. The decisions remain unchanged, and Invariants (1) and (2) hold after executing rule Unit.

We have $F \wedge \mathsf{decs}_{\leqslant n}(I) \models \overline{C \setminus \{\ell\}}$ and $F \wedge \mathsf{decs}_{\leqslant n}(I) \models C$, therefore, by modus ponens we get $F \wedge \mathsf{decs}_{\leqslant n}(I) \models \ell$. Since ℓ is not a decision literal, as shown above, $F \wedge \mathsf{decs}_{\leqslant n}(I\ell) \equiv F \wedge \mathsf{decs}_{\leqslant n}(I) \models I_{\leqslant n}$. Hence, $F \wedge \mathsf{decs}_{\leqslant n}(I\ell) \models (I\ell)_{\leqslant n}$, and Invariant (3) holds after executing rule Unit.

Jump: We first show that K contains no decision literal. In fact, the trail I is of the form $I = PQ$, and K is obtained from Q by removing all literals with decision level greater than b. In particular, the order of the remaining (decision) literals remains unaffected, and Invariant (1) still holds. We further have $\delta(K) \leqslant \delta(P) = b$. Since $\forall p \in P, k \in K \,.\, \tau(PK,p) < \tau(PK,k)$ and by the definition of decision literals in Equation 14.1, the decision literal with decision level b is contained in P. Therefore, since K contains no (decision) literal with decision level greater than b, it contains no decision literal.

Now we show that ℓ is not a decision literal either. As in the proof for rule Unit, it is sufficient to consider the case where $j > 0$. There exists a clause D where $F \models D$ such that $\delta(D) = c$ and a literal $\ell \in D$ for which $\ell|_Q = \bot$ and $\ell \in Q$. Since $j = \delta(D \setminus \{\ell\})$, $\delta(\ell) = \delta(D) = c > b$, and $\ell \notin K$. Instead, $\ell \in L$, and ℓ is unassigned during backtracking to any decision level smaller than c, i.e., $\ell \notin PK$. Furthermore, there exists a literal $k \in D$ where $k \neq \ell$ and such that $\delta(k) = j$ which precedes ℓ on the trail $PK\ell$. Therefore, following the argument in rule Unit, literal ℓ is not a decision literal, and since the decisions remain unchanged, Invariants (1) and (2) hold after applying rule Jump.

Invariant (3) holds prior to applying rule Jump, i.e., $F \wedge \mathsf{decs}_{\leqslant n}(I) \models I_{\leqslant n}$. We have that $F \models D$, and therefore $F \wedge D \equiv F$. Since $I = PQ$, $PK < I$ and obviously $F \wedge \mathsf{decs}_{\leqslant n}(PK) \implies (PK)_{\leqslant n}$. From $j = \delta(D \setminus \{\ell\})$ we get $D|_{PK} = \{\ell\}$. Repeating the argument in the proof for rule Unit by replacing I by PK and C by D, we have that $F \wedge \mathsf{decs}_{\leqslant n}(PQ\ell) \models (PQ\ell)_{\leqslant n}$, and Invariant (3) is met after executing rule Jump.

Decide: Literal ℓ is a decision literal by the definition of a decision literal in Equation 14.1: It is assigned decision level $d = \delta(I) + 1$, and $\forall k \in I \;.\; \delta(k) < \delta(\ell)$. Further, $\forall k \in I\ell \;.\; k \neq \ell \implies \tau(I\ell, k) < \tau(I\ell, \ell)$. Since $\ell \in \mathsf{decs}(I\ell)$, we have $\delta(\mathsf{decs}(I\ell)) = \{1, \ldots, d\}$, and Invariants (1) and (2) hold after applying rule Decide.

Since ℓ is a decision, $F \wedge \mathsf{decs}_{\leqslant n}(I\ell) \equiv F \wedge \mathsf{decs}_{\leqslant n}(I) \wedge \ell_{\leqslant n}$ and since Invariant (3) holds prior to applying Decide, obviously $F \wedge \mathsf{decs}_{\leqslant}(I\ell) \models I_{\leqslant n} \wedge \ell_{\leqslant n} \equiv (I\ell)_{\leqslant n}$, and Invariant (3) is met after applying rule Decide. $\square$

Proposition 14.1 (Correctness of Terminal State). Search *terminates in the correct state, i.e., if the terminal state is* SAT, *then F is satisfiable, and if the terminal state is* UNSAT, *then F is unsatisfiable.*

Proof. We show that the terminal state is correct for all terminal states.

SAT: We must show that an unsatisfiable formula can not be turned into a satisfiable one by any of the transition rules. Only equivalence-preserving transformations are executed: Rules Unit and Decide do not affect F, and in rule Jump a clause implied by F is added. Therefore, if the system terminates in state SAT, F is indeed satisfiable.

UNSAT: It must be proven that a satisfiable formula can not be made unsatisfiable. Only equivalence-preserving transformations are executed. Rules Unit and Decide do not affect F, and in rule Jump a clause implied by F is added. We need to show that if rule False is applied, the formula F is unsatisfiable. We have to consider Invariant (3) for $n = 0$. There exists a clause $C \in F$ such that $I_{\leqslant 0}(C) = \bot$, which leads to $F \wedge \mathsf{decs}_{\leqslant 0}(F) \equiv F \models I_{\leqslant 0}(C) = \bot$. $\square$

Proposition 14.2 (Progress and Termination). Search *makes progress in non-terminal states (a rule is applicable) and always reaches a terminal state.*

Proof. We first prove progress by showing that in every non-terminal state a transition rule is applicable. Then we prove termination by showing that no infinite state sequence is generated.

Progress: We show that in every non-terminal state a transition rule is applicable. The proof is by induction over the number of rule applications. Assume we reached a non-terminal state (F, I, δ). We show that one rule is applicable.

If $F|_I = \top$, rule True can be applied. If $F|_I = \bot$, there exists a clause $C \in F$ such that $C|_I = \bot$. The conflict level $\delta(C) = c$ may be either zero or positive. If $c = 0$, rule False is applicable. Now assuming $c > 0$ we obtain with Invariant (3):

$$F \wedge \mathsf{decs}_{\leqslant c}(I) \;\equiv\; F \wedge \mathsf{decs}_{\leqslant c}(I) \wedge I_{\leqslant c} \;\models\; I_{\leqslant c}.$$

Due to $I_{\leqslant c}(F) \equiv \bot$ we further have $F \wedge I_{\leqslant c} \equiv F \wedge \mathsf{decs}_{\leqslant c}(I) \equiv \bot$. By simply picking $\neg D = \mathsf{decs}_{\leqslant c}(I)$ we obtain $F \wedge \neg D \equiv F \wedge \neg D \wedge I_{\leqslant c} \equiv \bot$, thus $F \models D$. Clause D

contains only decision literals and $\delta(D) = c$. From Invariants (1) and (2) we know that D contains exactly one decision literal for each decision level in $\{1, \ldots, c\}$. We choose $\ell \in D$ such that $\delta(\ell) = c$. Then the asserting level is given by $j = \delta(D \setminus \{\ell\})$ and we pick some backtrack level b where $j \leqslant b < c$. Without loss of generalization we assume the trail to be of the form $I = PQ$ where $\delta(P) = b$. After backtracking to decision level b, the trail is equal to $I_{\leqslant b} = PK$ where $K = Q_{\leqslant b}$. Since $D|_{PK} = \{\ell\}$, all conditions of rule Jump hold.

If $F|_I \notin \{\top, \bot\}$, there are still unassigned variables in V. If there exists a clause $C \in F$ where $C|_I = \{\ell\}$, the preconditions of rule Unit are met. If instead $\mathsf{units}(F|_I) = \emptyset$, there exists a literal ℓ with $V(\ell) \in V$ and $\delta(\ell) = \infty$, and the preconditions of rule Decide are satisfied.

In this argument, all possible cases are covered and thus in any non-terminal state a transition rule can be executed, i.e., the system never gets stuck.

Termination: To show termination we follow the arguments in [124, 155] or more precisely the one in [28], except that our blocks (as formalized above with the block notion) might contain literals with different decision levels, i.e., subsequences of literals of the same decision level are interleaved as discussed in Section 14.3. This has an impact on the backtracking procedure adopted in rule Jump, where after backtracking to the end of $\mathsf{block}(I, b)$, trail P is extended by $K = Q_{\leqslant b}$ As discussed in the proof of Lemma 14.1, K contains no decision literals. Apart from that, the same argument applies as in [28], and SEARCH always terminates. □

14.6 ALGORITHM

The transition system presented in Section 14.4 can be turned into an algorithm described in Figure 14.4 providing a foundation for our implementation. Unlike in [149], we refrain from giving implementation details but provide pseudocode on a higher abstraction level covering exclusively chronological backtracking.

Search: The main function Search takes as input a formula F, a set of variables V, a trail I and a decision level function δ. Initially, I is equal to the empty trail and all variables are assigned decision level ∞.

If all variables are assigned and no conflict occurred, it terminates and returns SAT. Otherwise, unit propagation by means of Propagate is executed until either a conflict occurs or all units are propagated.

If a conflict at decision level zero occurs, Search returns UNSAT, since conflict analysis would yield the empty clause even if the trail contains literals with decision level higher than zero. These literals are irrelevant for conflict analysis (line 7), and they may be removed from I prior to conflict analysis without affecting the computation of the learned clause. The resulting trail contains only propagation literals, and the new (current) decision level is zero upon which the empty clause is learned.

If a conflict at a decision level higher than zero occurs, conflict analysis (function Analyze) is executed. If no conflict occurs and there are still unassigned variables, a decision is taken and a new block started.

Propagate: Unit propagation is carried out until completion. Unlike in CDCL with non-chronological backtracking, the propagated literals may be assigned a decision level lower than the current one (line 3). In this case invariant LevelOrder presented in Section 14.1 does not hold anymore. Propagate returns the empty clause if no conflict occurs and the conflicting clause otherwise.

Input: formula F, set of variables V, trail I, decision level function δ
Output: SAT iff F is satisfiable, UNSAT otherwise

```
Search ( $F$ )
1  $V := V(F)$
2  $I := \varepsilon$
3  $\delta := \infty$
4  **while** there are unassigned variables in $V$ **do**
5      $C :=$ *Propagate* ( $F, I, \delta$ )
6      **if** $C \neq \bot$ **then**
7          $c := \delta(C)$
8          **if** $c = 0$ **then return** UNSAT
9          *Analyze* ( $F, I, C, c$ )
10     **else**
11         *Decide* ( $I, \delta$ )
12 **return** SAT
```

```
Propagate ( $F, I, \delta$ )
1  **while** some $C \in F$ is unit $\{\ell\}$ under $I$ **do**
2      $I := I\ell$
3      $\delta(\ell) := \delta(C \setminus \{\ell\})$
4      **for all** clauses $D \in F$ containing $\neg\ell$ **do**
5          **if** $I(D) = \bot$ **then return** $D$
6  **return** $\bot$
```

```
Analyze ( $F, I, C, c$ )
1  **if** $C$ contains exactly one literal at decision level $c$ **then**
2      $\ell :=$ literal in $C$ at decision level $c$
3      $j := \delta(C \setminus \{\ell\})$
4  **else**
5      $D :=$ *Learn* ( $I, C$ )
6      $F := F \wedge D$
7      $\ell :=$ literal in $D$ at decision level $c$
8      $j := \delta(D \setminus \{\ell\})$
9  pick $b \in [j,\ c-1]$
10 **for all** literals $k \in I$ with decision level $> b$ **do**
11     assign $k$ decision level $\infty$
12     remove $k$ from $I$
13 $I := I\ell$
14 assign $\ell$ decision level $j$
```

Figure 14.4: This is the algorithm for CDCL with chronological backtracking, which differs from its non-chronological backtracking version as follows: Propagated literals ℓ are assigned a decision level which may be lower than the current one (line 3 in Propagate). The conflict level may be lower than the current decision level (line 7 in Search). If the conflicting clause contains only one literal at conflict level, it is used as reason and no conflict analysis is performed (lines 1–3 in Analyze). Picking the backtracking level is usually non-deterministic (line 9 in Analyze). Backtracking involves removing from the trail I all (literals in) slice(I, i) with $i > b$ (line 12 in Analyze).

Analyze: If the conflict level is higher than zero and the conflicting clause C contains exactly one literal ℓ at conflict level c, then C can be used as reason instead of performing conflict analysis (lines 1–3). The idea is to save the computational effort of executing conflict analysis and adding redundant clauses.

Otherwise, a clause D is learned as in CDCL, e.g., the first unique implication point (1st-UIP) containing exactly one literal ℓ at conflict level. Let j be the lowest decision level at which D (or C, if it contains exactly one literal at conflict level) becomes unit. Then according to some heuristics the solver backtracks to a decision level $b \in [j, c-1]$.

This for instance, can be used to retain part of the trail, to avoid redundant work which would repeat the same assignments after backtracking. Remember that the decision levels on the trail may not be in ascending order. When backtracking to b, the solver removes all literals with decision level higher than b from I, i.e., all i-th slices with $i > b$.

14.7 IMPLEMENTATION

We added chronological backtracking to our SAT solver CaDiCaL [19] based on the rules presented in Section 14.4, in essence implementing the algorithm presented in Figure 14.4, on top of a classical CDCL solver. This required the following four changes, similar to those described in [149] and implemented in the source code which was submitted by the authors to the SAT 2018 competition [150]. This list is meant to be comprehensive and confirms that the changes are indeed local.

Asserting Level. During unit propagation the decision level $a = \delta(C \backslash \{\ell\})$, also called asserting level, of propagated literals ℓ needs to be computed based on the decision level of all the falsified literals in the reason clause C. This is factored out in a new function called `assignment_level`[1], which needs to be called during assignment of a variable if chronological backtracking is enabled.

Conflict Level. At the beginning of the conflict analysis the current conflict level c is computed in a function called `find_conflict_level`[1]. This function also determines if the conflicting clause has one or more falsified literals on the conflict level. In the former case we can simply backtrack to backtrack level $b = c - 1$ and use the conflicting clause as reason for assigning that single literal on the conflict level. Even though not described in [149] but implemented in their code, it is also necessary to update watched literals of the conflict. Otherwise important low-level invariants are violated and propagation might miss falsified clauses later. In order to restrict the changes to the conflict analysis code to a minimum, it is then best to backtrack to the conflict level, if it happens to be smaller than the current decision level. The procedure for deriving the learned clause D can then remain unchanged (minimizing the 1st-UIP clause).

Backtrack Level. Then we select the backtrack level b with $j \leqslant b < c$, where j is the minimum backjump level j (the second largest decision level in D) in the function `determine_actual_backtrack_level`[1]. By default we adopted the heuristic from the original paper [149] to always force chronological backtracking ($b = c - 1$) if $c - j > 100$ (`T` in [149]) but in our implementation we do not prohibit chronological backtracking initially (`C` in [149]). Beside that we adopted a variant of reusing the trail [188] as follows. Among the literals on the trail assigned after and including

1 Please refer to the source code of CaDiCaL provided at http://fmv.jku.at/chrono.

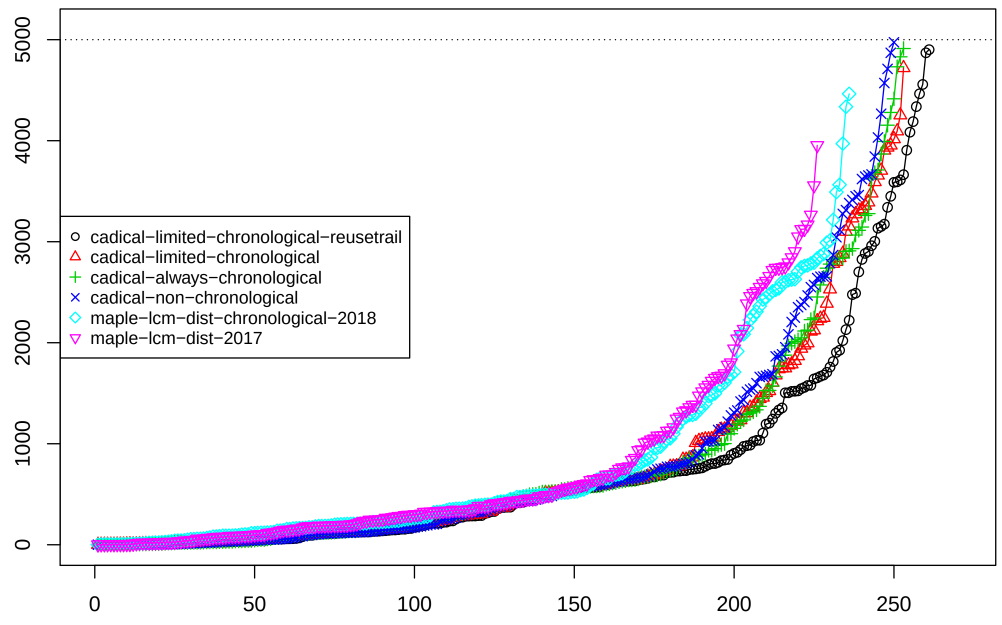


Figure 14.5: Cactus plot for benchmarks of the main track of the SAT Competition 2018.

the decision at level $j + 1$ we find the literal k with the largest variable score and backtrack to b with $b + 1 = \delta(k)$.

Flushing. Finally, the last required change was to flush literals from the trail with decision level larger than b but keep those smaller or equal than b. Instead of using an extra data structure (queue) as proposed in [149] we simply traverse the trail starting from block $b + 1$, flushing out literals with decision level larger than b. It is important to make sure that all the kept literals are propagated again (resetting the `propagated`[1] level).

14.8 EXPERIMENTS

We evaluated our implementation on the benchmarks from the main track of the SAT Competition 2018 and compare four configurations of CaDiCaL [19]. We also consider maple-lcm-dist-2017 [200], also called Maple_LCM_Dist, which won the main track of the SAT Competition 2017, on which maple-lcm-dist-chronological-2018 [150], also called Maple_LCM_Dist_ChronoBT, is based. We consider the latter as reference implementation for [149]. It won the main track of the SAT Competition 2018 on the considered benchmark set.

The experiments were executed on our cluster where each compute node has two Intel Xeon E5-2620 v4 CPUs running at 2.10 GHz with turbo-mode disabled. Time limit was set to 3600 seconds and memory limit to 7 GB. We used version "0nd" of CaDiCaL. Compared to the SAT Competition 2018 version [19] it incorporates new phase saving heuristics and cleaner separation between stabilizing and non-stabilizing phases [156]. This gave substantial performance improvements on satisfiable formulae [20]. Nevertheless adding chronological backtracking improves performance even further as the cactus plot in Figure 14.5 and Table 14.1 show.

The default version cadical-limited-chronological-reusetrail is best (preliminary experiments with CaDiCaL optimized for SAT Race 2019 did not confirm this

Table 14.1: Solved instances of the main track of the SAT Competition 2018.

solver configurations	solved instances		
	total	SAT	UNSAT
cadical-limited-chronological-reusetrail	261	155	106
cadical-limited-chronological	253	147	106
cadical-always-chronological	253	148	105
cadical-non-chronological	250	144	106
maple-lcm-dist-chronological-2018	236	134	102
maple-lcm-dist-2017	226	126	100

result though). It uses the limit $C = 100$ to chronologically backtrack if $c - j > 100$ and further reuses the trail as explained in the previous section. For cadical-limited-chronological reusing the trail is disabled and less instances are solved. Quite remarkable is that configuration cadical-always-chronological ranks third, even though it always enforces chronological backtracking ($b = c - 1$). On these benchmarks there is no disadvantage in always backtracking chronologically! The original classical CDCL variant cadical-non-chronological comes next followed by the reference implementation for chronological backtracking maple-lcm-dist-chronological-2018 and then maple-lcm-dist-2017 last, confirming the previous results in [149]. Source code and experimental data can be found at http://fmv.jku.at/chrono.

14.9 CONCLUSION

The success of MAPLE_LCM_DIST_CHRONOBT [150] is quite remarkable in the SAT Competition 2018, since the solver violates various invariants previously considered crucial for CDCL solvers (summarized in Figure 14.1). The corresponding paper [149] however was lacking proofs. In this paper we described and formalized a framework for combining CDCL with chronological backtracking. Understanding precisely which invariants are crucial and which are redundant was the main motivation for this paper. Another goal was to empirically confirm the effectiveness of chronological backtracking within an indepentendent implementation.

Our main contribution is to precisely define the concepts introduced in [149]. The rules of our framework simplify and generalize chronological backtracking. We may relax even more CDCL invariants without compromising the procedure's correctness. For instance first experiments show that during the application of the Unit rule it is not necessary to require that the formula is not falsified by the trail. Similarly, requiring the formula not to be falsified appears to be sufficient for rule Decide (no need to require that there are no units).

Our experiments confirm that combining chronological backtracking and CDCL has a positive impact on solver performance. We have further explored reusing the trail [188] during backjumping, which requires a limited form of chronological backtracking, too. Our experiments also show that performing chronological backtracking exclusively does not degrade performance much and thus for instance has potential to be used in propositional model counting. Furthermore, besides counting, possible applications may be found in SMT and QBF. We further plan to investigate the combination of these ideas with total assignments following [92].

15

DISCUSSION OF PAPER 3

In this chapter, we first recall the main contributions of the paper in Section 15.1. One particularity of chronological CDCL is that—unlike in CDCL with non-chronological backtracking—a conflict may occur at a decision level which is lower than the current one. In our rules, we do not specify how the conflict is analyzed and the conflict clause learnt, and in our implementation we backtrack to the conflict level before analyzing the conflict. However, this backtracking step is not needed from a technical point of view as discussed in Section 15.2. In the paper we mentioned that reusing the trail did not prove advantageous in combination with chronological CDCL after optimizing CaDiCaL for the SAT Race 2019. We precise the argument in Section 15.3 and provide the experimental results we could not include in the paper due to space limitations.

15.1 MAIN CONTRIBUTIONS

One goal was to investigate the suitability of chronological CDCL [149] in the context of propositional model counting (#SAT). Our intuition was that remaining in the proximity of the currently investigated search space also after a conflict, fewer assignments might be repeated saving redundant work.

We formulated invariants inherent to CDCL with non-chronological backtracking and identified those violated by backtracking chronologically after learning a conflict clause. Particularly the violation of invariant LevelOrder requires significant changes to the structure of the trail, since its literals are not ordered in ascending order with respect to their decision level anymore, and literals at the same decision level might occur in different blocks. Consequently, during backtracking, blocks are not discarded as a whole but only the literals at decision levels higher than the backtrack level are unassigned. Our slice-based trail structure provides a solution to this issue and facilitates referring to literals at the same decision level.

We formalized chronological CDCL. The rules capture termination with a conflict or a satisfying assignment, unit propagation, decisions, and CDCL with chronological backtracking. In the algorithm by Nadel and Ryvchin, backtracking after a conflict always occurs to the assertion level. In contrast, the jump level in our rule Jump can be freely chosen between and including the assertion level and the conflict level minus one, generalizing the method by Nadel and Ryvchin. We further provide a formal proof of correctness of our calculus.

Finally, we turned our rules into an algorithm providing the basis for their implementation in the SAT solver CaDiCaL [20], which won the SAT Track and became second in the SAT + UNSAT Track of the SAT Race 2019.[1] Since then, our rules

1 https://satcompetition.github.io/2019/

become an integrative part of our SAT solvers. The new SAT solver Kissat [22] won the Main Track UNSAT and the Main Track ALL and became second in the Main Track SAT and third in the Planning Track of the SAT Competition 2020.[2] The SAT solver Kissat [23] became second in the Main Track ALL and third in the Main Track UNSAT of the SAT Competition 2021.[3]

15.2 CONFLICT AT LOWER DECISION LEVEL

In our implementation, if a conflict occurs at a smaller decision level than the current one, we backtrack to the conflict level before analyzing the conflict. This enables us to use the same code for deriving the learnt clause independently of whether the conflict occurred at the maximum or a lower decision level. Technically, conflict analysis could also be executed without backtracking to the conflict level. The assignments at decision levels higher than the conflict level are not involved in the conflict and therefore neither in conflict analysis and can simply be ignored. But this is equivalent to backtracking to the conflict level prior to conflict analysis, and the learnt clause and the trail after executing rule Jump would be the same as shown in the following example.

Example 15.1 (Conflict analysis without backtracking to the conflict level)**.** *Consider the propositional formula*

$$F = \underbrace{(a \vee b)}_{C_1} \wedge \underbrace{(\neg b \vee c)}_{C_2} \wedge \underbrace{(\neg b \vee \neg d \vee e)}_{C_3} \wedge \underbrace{(\neg d \vee \neg f \vee g)}_{C_4} \wedge$$
$$\underbrace{(\neg a \vee \neg h \vee i)}_{C_5} \wedge \underbrace{(\neg a \vee \neg h \vee \neg i)}_{C_6} \wedge \underbrace{(\neg c \vee h \vee j)}_{C_7} \wedge \underbrace{(\neg c \vee h \vee \neg j)}_{C_8}$$

defined over the set of variables $V = \{a, b, c, d, e, f, g, h, i, j\}$. We represent the trail similarly to the paper, i. e., τ, I, and δ denote the trail position, the literals on the trail, and their decision level, respectively. Let the current trail be

τ	1	2	3	4	5	6	7	8	9
I	a^d	b^d	c^{C_2}	d^d	e^{C_3}	f^d	g^{C_4}	h^d	i^{C_5}
δ	1	2	2	3	3	4	4	5	5

The clause C_6 is falsified, and rule Jump is applied. The conflict level is $\delta(C_6) = 5$, which is the greatest decision level on the trail. Conflict analysis yields the clause $D_1 = (\neg a \vee \neg h)$ with $\delta(D_1) = 5$, which is added to F. The assertion level—called jump level in the paper—is one. The solver backtracks to decision level $\delta(C_6) - 1 = 4$ and propagates $\neg h$ with reason D_1 at decision level one. Now the literal j is propagated with reason C_7 and assigned decision level $\delta(C_7 \setminus \{j\}) = 2$. The resulting trail is given by

τ	1	2	3	4	5	6	7	8	9
I	a^d	b^d	c^{C_2}	d^d	e^{C_3}	f^d	g^{C_4}	$\neg h^{D_1}$	j^{C_7}
δ	1	2	2	3	3	4	4	1	2

The literals $\neg h$ and k, marked in red, are out of order: we have $\tau(I, g) = 7 < 8 = \tau(I, \neg h)$ but $\delta(g) = 4 > 1 = \delta(\neg h)$ and $\tau(I, g) = 7 < 9 = \tau(I, j)$ and $\delta(g) = 4 > 2 = \delta(j)$. Again, a conflict occurs. The conflicting clause is C_8 with $\delta(C_8) = 2 < 4 = \delta(I)$.

2 https://satcompetition.github.io/2020/
3 https://satcompetition.github.io/2021/

CONFLICT ANALYSIS WITHOUT BACKTRACKING TO CONFLICT LEVEL. *Suppose we execute rule* Jump *without backtracking to the conflict level before analyzing the conflict. The implication graph [127] is very useful in reading off the resolution steps executed during conflict analysis. It is a directed acyclic graph whose nodes represent the assigned literals and incoming edges represent propagations and are labeled with the reason of the literal represented by the node they are pointing to. Decision nodes have no incoming edge, and the special node κ represents a conflict. The implication graph corresponding to our example is shown below on the left hand side.*

$$\begin{tikzcd}
a \arrow[r, dashed, "D_1"] & \neg h \arrow[dr, dashed, "C_7"] \arrow[drr, dashed, bend left, "C_8"] & & \\
b \arrow[r, "C_2"] \arrow[dr, "C_3"] & c \arrow[r, dashed, "C_7"] \arrow[rr, dashed, bend right, "C_8"'] & j \arrow[r, dashed, "C_8"] & \kappa \\
d \arrow[r, "C_3"] \arrow[dr, "C_4"] & e & & \\
f \arrow[r, "C_4"] & g & &
\end{tikzcd}$$

$$\frac{(\neg c \vee h \vee \neg j) \qquad (\neg c \vee h \vee j)}{(\neg c \vee h)}$$

The out-of-order propagations are highlighted in red and dashed. The computation of the conflict clause occurs by traversing the implication graph in reverse assignment order and is shown on the right hand side: the conflicting clause C_8 is resolved with C_7, the reason of j. The resolvent $(\neg c \vee h)$ contains one single literal at decision level two, namely $\neg c$, and we add $D_2 = (\neg c \vee h)$ to F. The assertion level is one, the second highest decision level in D_2, and backtracking occurs to conflict level minus one, which is one. Backtracking to decision level one involves unassigning all literals at decision levels greater than one. The resulting trail is $a^d \neg c^{D_1}$, under which the residual of D_2 becomes the unit $(\neg c)$. After propagating $\neg c$ with reason D_2, the trail becomes

τ	1	2	3
I	a^d	$\neg h^{D_1}$	$\neg c^{D_2}$
δ	1	1	1

All literals on the trail are now in order. However, this is a particularity of this example and need not be the case after executing rule Jump *in general.*

CONFLICT ANALYSIS AFTER BACKTRACKING TO CONFLICT LEVEL. *Now consider again the situation above right after propagating j with reason C_7, where a conflict in C_8 is obtained but let us backtrack to the conflict level two before analyzing the conflict. The resulting trail contains only literals at decision levels one and two:*

τ	1	2	3	4	5
I	a^d	b^d	c^{C_2}	$\neg h^{D_1}$	j^{C_7}
δ	1	2	2	1	2

The corresponding implication graph and the resolution step for determining the conflict clause are shown below on the left and right hand side, respectively.

$$\begin{tikzcd}
a \arrow[r, dashed, "D_1"] & \neg h \arrow[dr, dashed, "C_7"] \arrow[rrd, dashed, bend left, "C_8"] & & \\
b \arrow[r, "C_2"] & c \arrow[r, dashed, "C_7"] \arrow[rr, dashed, bend right, "C_8"'] & j \arrow[r, dashed, "C_8"] & \kappa
\end{tikzcd}$$

$$\frac{(\neg c \vee h \vee \neg j) \qquad (\neg c \vee h \vee j)}{(\neg c \vee h)}$$

The implication graph is precisely the upper part of the implication graph shown above. Since the lower part of this implication graph was not involved in the conflict, the same conflict clause is obtained independently of whether backtracking to the conflict level occurred prior to conflict analysis or not. The resolution step executed for computing D_2 is shown on the right hand side and coincides with the one obtained previously.

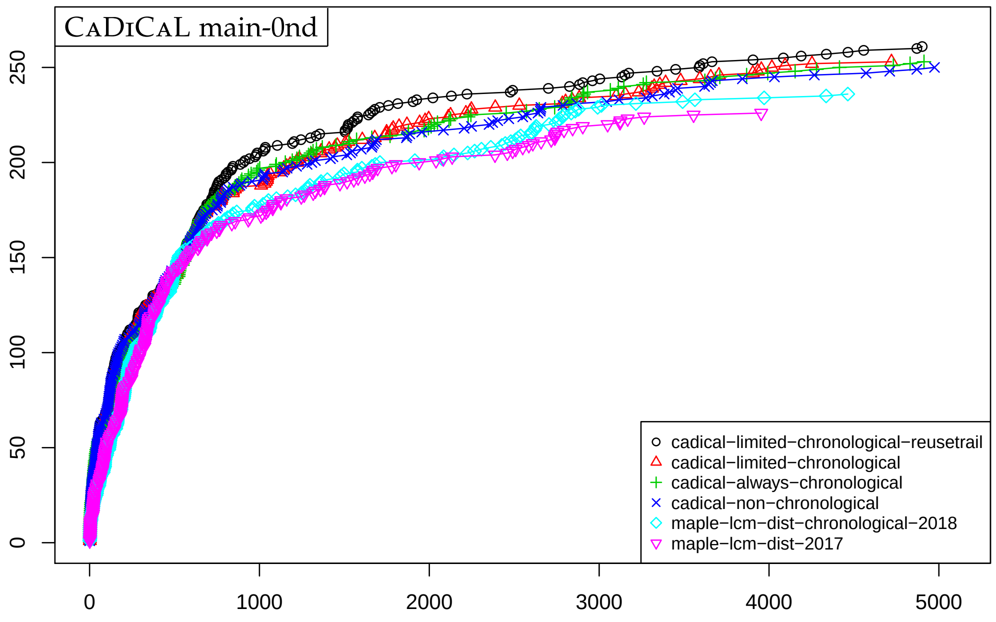
15.3 THE IMPACT OF REUSING THE TRAIL

In the experiments reported in the paper, among the configurations of CaDiCaL supporting chrononological CDCL, the best solver performance was achieved in combination with reusing the trail [188] during chronological backtracking, as shown in the cumulative distribution function (CDF) plot corresponding to the cactus plot in the paper and visualized in Figure 15.1 above. The x-axis denotes the execution time in seconds and the y-axis denotes the number of solved instances. The higher a curve, the better the performance, and the upper left corner is best.

However, in the same experiments conducted with CaDiCaL optimized for the SAT Race 2019, reusing the trail turned out to be not that useful but not very harmful, either. In fact, cadical-limited-chronological-reusetrail performed even worse than cadical-non-chronological, as can be read off the CDF plot in Figure 15.1 below: the configuration cadical-limited-chronological performed best among all CaDiCaL configurations but worst if combined with reusing the trail. In the rest of this section, we focus on the results for CaDiCaL submitted to the SAT Competition 2018 and CaDiCaL optimized for SAT Race 2019.

The results for the version of CaDiCaL optimized for SAT Race 2019 are given in Table 15.1 with the numbers in brackets denoting the difference to the version of the paper. Considering that cadical-limited-chronological and cadical-always-chronological solved more instances than cadical-non-chronological, it looks like reusing the trail in combination with chronological CDCL did even harm solver performance. Comparing the number of instances solved by cadical-limited-chronological and cadical-limited-chronological-reusetrail is particularly interesting, since the latter differs from the former only in reusing the trail. Adding it to cadical-limited-chronological resulted in solving ten fewer instances, four satisfiable and six unsatisfiable ones. While cadical-limited-chronological improved most after optimization for SAT Race 2019, and the other configurations achieved a significant increase in the number of solved instances as well, cadical-limited-chronological-reusetrail improved least, even less than cadical-non-chronological.

The last column in Table 15.1 refers to the number of instances solves exclusively by the corresponding solver configuration. Except for cadical-limited-chronological-reusetrail and the Maple configurations, these numbers increased as well. Notice the significant decrease for cadical-limited-chronological-reusetrail, which is higher than the increase for cadical-limited-chronological.



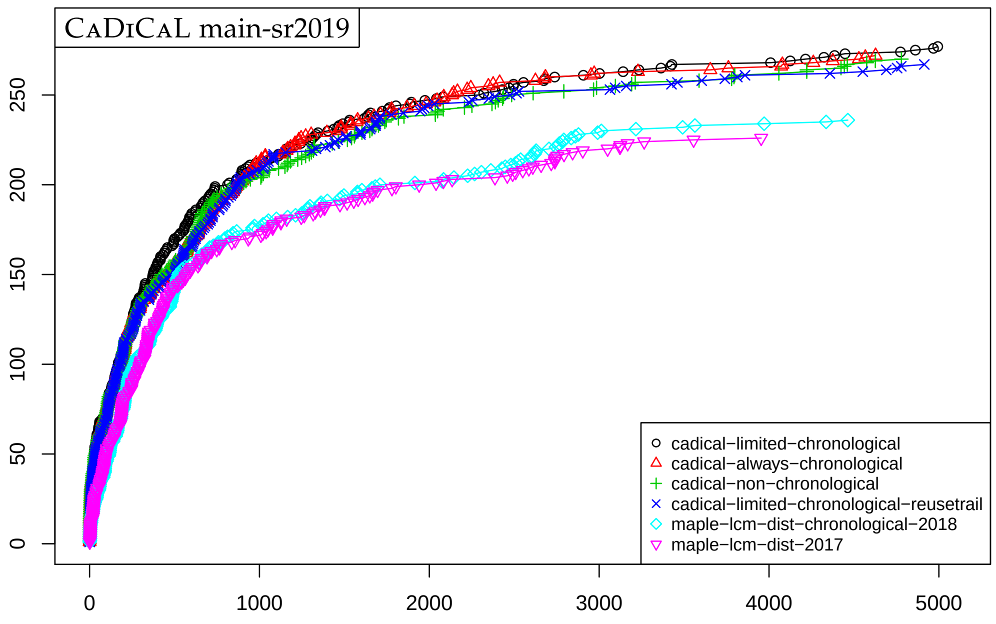


Figure 15.1: Cumulative distribution function (CDF) plot visualizing the number of solved instances of the main track of the SAT Competition 2018 by the version of CaDiCaL in the paper (above) and optimized for SAT Race 2019 (below).

Table 15.1: Number of solved instances in the main track of the SAT Competition 2018 by CaDiCaL optimized for the SAT Race and in the paper (in brackets).

SOLVER CONFIGURATIONS	SOLVED INSTANCES			
	TOTAL	SAT	UNSAT	UNIQ
cadical-limited-chronological	277 (+24)	164 (+17)	113 (+7)	4 (2)
cadical-always-chronological	272 (+19)	163 (+15)	109 (+4)	2 (2)
cadical-non-chronological	270 (+20)	160 (+16)	110 (+4)	1 (0)
cadical-limited-chronological-reusetrail	267 (+6)	160 (+5)	107 (+1)	1 (7)
maple-lcm-dist-chronological-2018	236	134	102	4 (3)
maple-lcm-dist-2017	226	126	100	1 (0)

The order with respect to the number of solved instances of the configurations of CaDiCaL without reusing the trail remained the same: cadical-limited-chronological performed best followed by cadical-always-chronological and cadical-non-chronological. Notably, the percentage increase of the number of solved instances amounts to 7.51% for cadical-always-chronological, 8.00% for cadical-non-chronological, and 9.49% for cadical-limited-chronological, but only to 2.30% for cadical-non-chronological. This confirms the positive impact on solver performance of chronological backtracking for the instances of the main track of the SAT Competition 2018, but raises questions concerning reusing the trail.

16

PAPER 4: COMBINING CONFLICT-DRIVEN CLAUSE LEARNING AND CHRONOLOGICAL BACKTRACKING FOR PROPOSITIONAL MODEL COUNTING

PUBLISHED. In: *GCAI 2019. Proceedings of the 5th Global Conference on Artificial Intelligence, Bozen/Bolzano, Italy, 17–19 September 2019*. Ed. by Diego Calvanese and Luca Iocchi. Vol. 65. EPiC Series in Computing. EPiC Series in Computing. EasyChair, 2019, pp. 113–126. DOI: 10.29007/vgg4.

AUTHORS. Sibylle Möhle and Armin Biere.

ACKNOWLEDGMENTS. Supported by the Austrian Science Fund (FWF) grant S11408-N23 (RiSE) and by the LIT Secure and Correct Systems Lab funded by the State of Upper Austria.

ABSTRACT. In propositional model counting, also named #SAT, the search space needs to be explored exhaustively, in contrast to SAT, where the task is to determine whether a propositional formula is satisfiable. While state-of-the-art SAT solvers are based on non-chronological backtracking, it has also been shown that backtracking chronologically does not significantly degrade solver performance. Hence investigating the combination of chronological backtracking with conflict-driven clause learning (CDCL) for #SAT seems evident. We present a calculus for #SAT combining chronological backtracking with CDCL and provide a formal proof of its correctness.

16.1 INTRODUCTION

The task of computing the number of models of a propositional formula, also referred to as #SAT, has various applications in hardware and software verification [26, 36, 73, 74, 104] as well as cryptography [106]. Classical applications are found in the area of probabilistic reasoning [167] as well as Bayesian networks [13, 120, 174] which are adopted in medical diagnosis and planning. A counting characterization of diagnoses is presented in [112]. Propositional model counting finds further applications in product configuration [110, 203] and planning [10, 202].

Challenges in model counting. In contrast to SAT solving, where the search terminates as soon as a satisfying variable assignment is found, in #SAT the search space needs to be explored exhaustively. State-of-the-art SAT solvers implement conflict-driven clause learning (CDCL) [128, 146], i.e., in case of a conflict they learn a clause which might be used to *backjump* or *backtrack non-chronologically* to a potentially much smaller decision level by unassigning all literals at greater decision levels. This allows them to escape search space regions without solution. In the context of #SAT, however, backjumping might cause an erroneous model count since already counted models might be found again.

This problem does not occur in the Davis-Putnam-Logeman-Loveland (DPLL) algorithm [60] where only *chronological backtracking*, i.e., backtracking to the previous decision level, is applied. However, the downside of DPLL consists of the fact that the solver is not able to escape regions of the search space without solution easily.

One therefore might ask whether chronological backtracking and CDCL might be combined in order to benefit from the strengths of both methods. A second motivation is given by the fact that CDCL may lead to redundant work since after backjumping assignments discarded might be repeated [188] which has a greater negative impact on solver performance in #SAT.

Combining CDCL and chronological backtracking. A partial answer to this question was provided in [149]. In this paper, chronological backtracking was enabled under certain conditions violating invariants considered crucial to CDCL. In CDCL with non-chronological backtracking the current variable assignment is represented by the trail which is a sequence of literals. The trail is partitioned into subsequences of literals between decisions identified by a decision level which is assigned to the literals contained in them. One of the violated invariants says that the literals on the trail are ordered in ascending order with respect to their decision level.

This is not the case in [149] anymore. Assume the formula under consideration evaluates to false under the current assignment. A clause is learned which is used to determine the jump level, i.e., the decision level to which the solver backjumps. Let ℓ be the literal of the learned clause propagated after backjumping. It is assigned to the jump level which, after backtracking chronologically to the backtrack level, might be lower than the decision level of another literal preceding ℓ on the trail. In this case variable assignments at decision levels higher than the jump level might conflict with the current assignment. This issue is dealt with by introducing a variant of backtracking in which no blocks of literals are unassigned but only the literals with decision level greater than the backtrack level, which need not be consecutive anymore.

To define more precisely the concepts introduced in [149], in [138] we provided a formalization and formal proof and generalization of the procedure. Our goal was to understand which invariants are crucial and which are redundant and to empirically confirm the effectiveness of chronological backtracking. A second motivation, which is explored in depth in this paper, was the potential use of chronological backtracking for model counting, i.e., #SAT.

In [138], we characterized five invariants, three of which are violated by chronological backtracking. Still the procedure remains correct and we proved empirically that always executing chronological backtracking in combination with CDCL does not degrade significantly solver performance confirming the suitability of chronological backtracking for model counting.

Exact propositional model counting. The state of the art in exact propositional model counting goes back to a paradigm presented in [15]. In this method the formula under consideration is split into subformulae over disjoint sets of variables which are then solved independently and their results multiplied to yield the model count of the formula. In this work the need for so-called *good learning* is identified. Several modern #SAT solvers implement this paradigm [172, 173, 189], a parallel version [37] as well as a distributed version [38] are available. Another method is based on knowledge compilation [59] in which the formula is transformed into another language in which model counting can be executed efficiently.

A totally different approach, inspired by [72, 93], was presented in [24, 137]. It is *dual*, i.e., it takes as input a formula together with its negation. The basic idea is based on the fact that a formula evaluates to $\top$ under a given variable assignment if and only if its negation evaluates to $\bot$ under the same assignment. This method also enables the detection of partial models.

Our contribution. In this work we extend our framework [138] to serve propositional model counting and provide a formal proof of its correctness. We present a counting algorithm based on model enumeration. Basically, our procedure yields a formula which is logically equivalent to the formula under consideration. The generated formula is a disjunction of pairwise contradicting conjunctions of literals, hence its model count can be determined in polynomial time.

While model enumeration and model counting are two different yet related tasks, our method is based on the first one since it facilitates reasoning about the method's properties in the following manner. Our argument is based on the concepts of *pending search space*, i.e., the variable assignments not yet tested, and *pending models*, i.e., the models not yet found. Obviously the two concepts are related, and the pending models are contained in the pending search space. In an implementation therefore one might decide to sum up the number of the detected models instead of recording them explicitly.

First, we extend the original calculus [138] by one rule, to capture the situation where a model is found. Second, the states now include all the (partial) models found at any point in time of the search. Compared to [137] this extension is not dual. It also exclusively uses chronological backtracking which renders blocking clauses superfluous. As a consequence, it does not need the concepts of *flipping* nor *discounting*, as introduced in [137]. In [137] we also showed by an example that combining dual reasoning and the use of blocking clauses might result in models counted twice. We also provided another example in which combining discounting and blocking clauses leads to the loss of models, i.e., a discounted model m is not detected anymore due to a blocking clause. These issues do not exist anymore in our new framework.

16.2 PRELIMINARIES

Let F be a propositional formula over the set of variables V. We call *literal* a variable $v \in V$ or its negation $\neg v$. The variable of a literal ℓ is obtained by $V(\ell)$. We denote with $\overline{\ell}$ the complement of the literal ℓ, i.e., $\overline{\ell} = \neg\ell$, supposing $\neg\neg\ell = \ell$.

A *clause* is a disjunction of literals, and a *cube* is a conjunction of literals. A formula in *Conjunctive Normal Form (CNF)* is a conjunction of clauses, whereas a *Disjoint Sum-of-Products (DSOP)* formula is a disjunction of pairwise contradicting cubes. We interpret CNF formulae as sets of clauses and DSOP formulae as sets of cubes where convenient. By writing $C \in F$ we refer to a clause or cube C in a formula F. Analogously we write $\ell \in C$ if ℓ is a literal in a clause or cube C. The empty CNF formula and the empty cube are represented by $\top$, the empty DSOP formula and the empty clause are denoted by $\bot$.

A *trail*, written $I = \ell_1 \dots \ell_n$, is a sequence of literals with no duplicate variables which may also be interpreted as the conjunction of its literals. By $V(I)$ the vari-

τ	0	1	**2**	3	**4**
I	1	2	**3**	4	**5**
δ	0	1	**0**	2	**1**

τ	0	1	**2**	3
I	1	2	**3**	5
δ	0	1	**0**	1

τ	0	1
I	1	3
δ	0	0

Figure 16.1: In the trail I on the left, from the three trails shown, literals 3 and 5 are placed out of order. In fact, their decision level δ is lower than the decision level of a literal preceding them on the trail, i.e., with lower position τ. The trails in the middle and on the right show the results of backtracking to decision levels 1 and 0. When backtracking to the *backtrack level* b, only literals ℓ with $\delta(\ell) > b$ are removed from the trail, while the assignment order is preserved.

ables of the literals on the trail I are obtained. We denote the empty trail by ε. Trails may be concatenated, $I = JK$, provided $V(J) \cap V(K) = \emptyset$. We write $J \leqslant I$ if J is a subsequence of I and $J < I$ if furthermore $J \neq I$. The position of literal ℓ on the trail I is denoted by $\tau(I, \ell)$. We may interpret I as a set of literals and write $\ell \in I$ to refer to the literal ℓ on I. Similarly to [137], the decision literals on the trail are marked by a superscript, e.g., ℓ^d denoting open "left" branches in the sense of DPLL. Flipping the value of a decision can be seen as closing the corresponding left branch and starting the "right" branch. Thus its decision literal ℓ^d becomes a *flipped literal* $\overline{\ell}$.

The *decision level function* $\delta\colon V \mapsto \mathbb{N} \cup \{\infty\}$ returns the decision level of a variable $v \in V$. We define $\delta(v) = \infty$ if v is unassigned, and δ is updated whenever a variable is assigned or unassigned. The extension to literals ℓ, clauses C and trails I is straightforward by defining $\delta(\ell) = \delta(V(\ell))$, $\delta(C) = \max\{\delta(\ell) \mid \ell \in C\}$ for $C \neq \bot$ and $\delta(I) = \max\{\delta(\ell) \mid \ell \in I\}$ for $I \neq \varepsilon$. We further define $\delta(\bot) = \delta(\varepsilon) = 0$. We write $\delta[\ell \mapsto d]$ to denote the updated function δ in which $V(\ell)$ is assigned to decision level d. Similarly, by $\delta[I \mapsto \infty]$ all literals on the trail I are unassigned. We may write $\delta \equiv \infty$ as a shortcut. The function δ is left-associative, i.e., $\delta[I \mapsto \infty][\ell \mapsto e]$ first unassigns all literals on I and then assigns literal ℓ to decision level e.

By $\delta(L) = \{\delta(\ell) \mid \ell \in L\}$ we denote the set containing the decision levels of its elements. The set of decision literals on the trail I is denoted by $\mathsf{decs}(I) = \{\ell \mid \ell^d \in I\}$ and the set containing their complements by $\overline{\mathsf{decs}(I)} = \{\overline{\ell} \mid \ell \in \mathsf{decs}(I)\}$. By $I_{\leqslant n}$ we refer to the subsequence of I containing only the literals $\ell \in I$ where $\delta(\ell) \leqslant n$. Analogously, the set $\mathsf{decs}(I)$ may be restricted to decision literals ℓ where $\delta(\ell) \leqslant d$ by writing $\mathsf{decs}_{\leqslant d}(I) = \mathsf{decs}(I_{\leqslant d})$. We call *slice* a subsequence of I containing all literals in I having the same decision level, e.g., $\mathsf{slice}(I, n) = I_{=n}$. Backtracking to decision level i now amounts to unassigning literals $\ell \in I$ with decision level greater than i, i.e., where $\ell \in \mathsf{slice}(I, j)$ for $j > i$.

We clarify these concepts by means of an example taken from [138]. Consider the trail I on the left hand side of Figure 16.1 over variables $\{1, \ldots, 5\}$ (in DIMACS format) where τ represents the position of a literal on I and δ represents its decision level. Literals 1 and 3 were propagated at decision level zero, literal 5 was propagated at decision level one. The literals 3 and 5 are *out-of-order literals*, i.e., their decision level is smaller than the one of a literal preceding them: We have $\delta(2) = 1 > 0 = \delta(3)$, whereas $\tau(I, 2) = 1 < 2 = \tau(I, 3)$. In a similar manner, $\delta(4) = 2 > 1 = \delta(5)$, and $\tau(I, 4) = 3 < 4 = \tau(I, 5)$. Moreover, $I_{\leqslant 1} = 1\,2\,3\,5$, $\mathsf{decs}(I) = \{2, 4\}$, $\mathsf{decs}_{\leqslant 1}(I) = \{2\}$, $\mathsf{slice}(I, 1) = 2\,5$. Upon backtracking to decision level one, the literals in $\mathsf{slice}(I, 2)$ are unassigned. The resulting trail is visualized in the middle of Figure 16.1. Note that since the assignment order is preserved, the trail still contains one out-of-order literal, namely 3. Backtracking to decision level zero unassigns all literals in $\mathsf{slice}(I, 2)$ and $\mathsf{slice}(I, 1)$ resulting in the trail in which all literals are placed *in order*, i.e., they are ordered in ascending order with respect to their decision level, depicted on the right hand side.

A *total assignment* is a mapping from the set of variables V to the truth values 0 and 1. The trail I may also be interpreted as a *partial assignment* where $I(\ell) = 1$ iff $\ell \in I$ and $I(\ell) = 0$ iff $\neg\ell \in I$. Thus, $I(\ell)$ is undefined if $V(\ell) \notin V(I)$. Analogously, $I(C)$ and $I(F)$ are defined. We denote with $V - I = V \setminus V(I) \subseteq V$ the set of variables in V and not in I. By $2^{|V-I|}$ we denote the number of total assignments covered by the partial assignment represented by I.

The *residual* of a formula F under the trail I is referred to by $F|_I$. It is obtained by replacing the literals $\ell \in I$ in F by $\top$ and their negation with $\bot$. Analogously, $C|_I$ for a clause C is defined. As an example, let $F = (a \vee b) \wedge (\overline{b} \vee c)$ be a formula and $I = a\, b$ the current trail. Then the residual of F under I is given by $F|_I = (c)$.

We say that the trail I *satisfies* a formula F, denoted by $I \models F$, if $I(F) \equiv \top$, i.e., $F|_I = \top$. Then I is called a *model* of F. The *model count* of F, denoted by $\#F$, is given by the number of total assignments satisfying F. If instead $I(F) \equiv \bot$, i.e., $I(C) \equiv \bot$ for a clause $C \in F$, we say that I *falsifies* F, call C the *conflicting clause* and $\delta(C)$ the *conflict level*.

A *unit clause* is a clause $\{\ell\}$ containing one single literal ℓ called *unit literal*. By $\mathsf{units}(F)$ we refer to the set of unit literals in F. Similarly, $\mathsf{units}(F|_I)$ is defined. By writing $\ell \in \mathsf{units}(F|_I)$ we denote that the unit clause $\{\ell\}$ is contained in the residual of F under I.

16.3 COUNTING VIA ENUMERATION WITH CHRONOLOGICAL CDCL

Let F be a CNF formula over variables V. A DSOP representation M of F consists of cubes representing pairwise disjoint sets of (partial) models of F. Obviously, M is not unique. The model count of M and hence of F equals the sum of the model counts of the cubes in M:

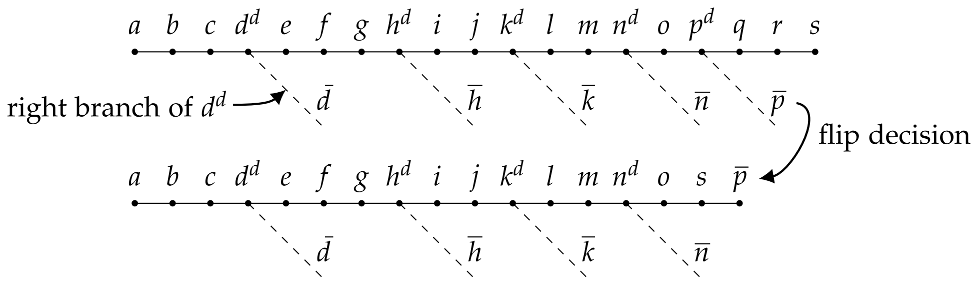
$$M \text{ is a DSOP representation of } F \implies \#F = \sum_{C \in M} 2^{|V-C|}$$

Let I be the current trail. We denote with $O(I)$ the *pending search space of* I, i.e., the assignments consistent with the trail not yet tested. It is given by I and its *(open) right branches* $R(I)$. Obviously $O(I)$ is not known, but it can be determined from I. Let I be of the form $I = I_0 \ell_1^d I_1 \ldots \ell_m^d I_m$. Then the pending search space with respect to the trail I is given by

$$\begin{aligned} O(I) &= \overline{\ell_1} \wedge I_{<1} \vee \overline{\ell_2} \wedge I_{<2} \vee \overline{\ell_3} \wedge I_{<3} \vee \ldots \vee \overline{\ell_m} \wedge I_{<m} \vee I_{\leqslant m} \\ &= I_{=0} \wedge (\overline{\ell_1} \vee I_{=1} \wedge (\overline{\ell_2} \vee I_{=2} \wedge (\overline{\ell_3} \vee I_{=3} \wedge (\ldots \vee I_{=m-1} \wedge (\overline{\ell_m} \vee I_{=m}) \ldots)))) \end{aligned}$$

which, if multiplied out, obviously is a DSOP (since $I_{=i}$ contains ℓ_i). Its cubes represent pairwise disjoint sets of total assignments. More generally, we define the pending search space of a trail I as

$$\begin{aligned} O(I) &= I \vee R(I) \quad \text{where} \\ R(I) &= \bigvee_{\ell \in \mathsf{decs}(I)} R_{=\delta(\ell)}(I) \quad \text{and} \quad R_{=\delta(\ell)}(I) = \overline{\ell} \wedge I_{<\delta(\ell)} \quad \text{for } \ell \in \mathsf{decs}(I) \end{aligned}$$



a b c d^d e f g h^d i j k^d l m n^d o p^d q r s
right branch of d^d
$\bar{d}$ $\bar{h}$ $\bar{k}$ $\bar{n}$ $\bar{p}$
flip decision
a b c d^d e f g h^d i j k^d l m n^d o s $\bar{p}$
$\bar{d}$ $\bar{h}$ $\bar{k}$ $\bar{n}$

Figure 16.2: Trail after backtracking chronologically in Back{True, False}. The trail itself together with the dashed lines represents the pending search space, i.e., the assignments not yet tested.

denotes the right branch of the decision ℓ at decision level $\delta(\ell)$. We also define $R_{\mathsf{op}\,n}(I) = R(I_{\mathsf{op}\,n})$, where $\mathsf{op} \in \{\leqslant, \geqslant, <, >, =\}$. Then the *pending models* of F, i.e., the models of F contained in $O(I)$, are given by $F \wedge O(I)$. Assuming the models of F already found are represented by M, then $M \vee (F \wedge O(I)) \equiv F$. The pending search space $O(I)$ and the found models M are both DSOPs as well as their disjunction $M \vee O(I)$, assuming $M \wedge O(I) \equiv \bot$. The model count of F therefore is given by

$$\#F = \#(F \wedge O(I)) + \sum_{C \in M} 2^{|V-C|}$$

Example 16.1. *Let F be a formula and let the trail be $I = abcd^defgh^dijk^dlmn^dop^dqrs$ with decisions $\mathsf{decs}(I) = \{d, h, k, n, p\}$ where $\delta(d) = 1$, $\delta(h) = 2$, $\delta(k) = 3$, $\delta(n) = 4$, and $\delta(p) = 5$. Let the further decision levels be $\delta(a) = \delta(b) = \delta(c) = \delta(f) = 0$, $\delta(e) = \delta(g) = \delta(m) = 1$, $\delta(i) = \delta(j) = \delta(l) = 2$, $\delta(s) = 3$, $\delta(o) = 4$, and $\delta(q) = \delta(r) = 5$. The slices are $I_{=0} = abcf$, $I_{=1} = d^degm$, $I_{=2} = h^dijl$, $I_{=3} = k^ds$, $I_{=4} = n^do$, $I_{=5} = p^dqr$. We have*

$$\begin{aligned} O(I) &= I \vee \bigvee_{\ell \in \mathsf{decs}(I)} (\bar{\ell} \wedge I_{<\delta(\ell)}) \\ &= I \vee (\bar{d} \wedge I_{<1}) \vee (\bar{h} \wedge I_{<2}) \vee (\bar{k} \wedge I_{<3}) \vee (\bar{n} \wedge I_{<4}) \vee (\bar{p} \wedge I_{<5}) \end{aligned}$$

visualized in Figure 16.2. The dashed lines below the decision literals denote the assignments where the respective decision literal is flipped, i.e., together with I they form the pending search space.

Now assume $I(F) = \top$. We assume we backtrack chronologically and flip the last decision literal p. We get $J = abcd^defgh^dijk^dlmn^dos\bar{p} = I_{<5}\bar{p}$ as shown at the bottom of Figure 16.2 and

$$\begin{aligned} O(J) &= J \vee \bigvee_{\ell \in \mathsf{decs}(J)} (\bar{\ell} \wedge I_{<\delta(\ell)}) \\ &= (\bar{d} \wedge I_{<1}) \vee (\bar{h} \wedge I_{<2}) \vee (\bar{k} \wedge I_{<3}) \vee (\bar{n} \wedge I_{<4}) \vee (\bar{p} \wedge I_{<5}) \end{aligned}$$

where $\delta(\bar{p}) = 4$. Note that $O(J) \vee I = O(I)$.

We have $O(I) = O(J) \vee I$. Thus $F \wedge O(I) = F \wedge O(J) \vee F \wedge I = F \wedge O(J) \vee I$ since $I \models F$ and the pending models of F with respect to J are exactly the pending models of F with respect to I minus the models of F represented by I.

16.4 CALCULUS

We describe our calculus as a state transition system where non-terminal states are denoted by (F, I, M, δ) in which F is a CNF formula over variables V, I denotes the current trail and M the DSOP containing the models found so far. Finally, δ denotes the decision level function. The initial state is given by $(F, \varepsilon, \bot, \delta_0)$ where F, ε and $\bot$ denote the original formula, the empty trail and the empty DSOP and $\delta_0 \equiv \infty$. The terminal state is equal to M which is a DSOP representation of F. The rules are listed in Figure 16.3. Next we provide explanations of the rules, and in Section 16.5 we proof their correctness.

EndTrue / EndFalse. If the trail I satisfies F and there are no decision literals left on I, it is added to M and the search terminates since the search space has been traversed exhaustively. If the trail I falsifies F, there exists a clause $C \in F$ such that I falsifies C. If in addition the conflict level $\delta(C)$ is zero, the pending search space is given by $O(I) = I_{\leqslant 0}$. We have $I_{\leqslant 0}(F) = \bot$, i.e., there are no more assignments to be tested, and the search terminates.

Unit. The trail I neither satisfies nor falsifies F and there exists a clause $C \in F$ that becomes a unit clause $\{\ell\}$ under I. The trail I is extended by ℓ which is assigned to decision level $\delta(C \setminus \{\ell\})$.

BackTrue. The trail I is a model of F and is added to M. We define D as the clause consisting of all negated decision literals on I. There exist subsequences P and Q of I such that $I = PQ$ and $\delta(P) = \delta(I) - 1 = e$ and a literal $\ell \in D$ such that $D|_P = \{\ell\}$. We use D as a kind of "reason" and backtrack chronologically to decision level e. This involves unassigning all literals with decision level greater than e. For instance, $\overline{\ell} \in \mathsf{decs}(I)$ and $\delta(\overline{\ell}) = e + 1$ and upon backtracking $\overline{\ell}$ is unassigned. After backtracking we propagate ℓ and assign it decision level $\delta(D \setminus \{\ell\}) = e$ resulting in the trail $PK\ell$ where $\delta(PK\ell) = e$.

BackFalse. There exists a clause $C \in F$ such that $C|_I = \bot$. The conflict level is $\delta(C) > 0$, i.e., there is a decision left on I. We can determine (usually by way of conflict analysis[1] or just using the negated decisions) a clause D for which $F \wedge \overline{M} \models D$ such that $\delta(D) = c > 0$ and whose residual is unit, e.g., $\{\ell\}$, at jump level $j = \delta(D \setminus \{\ell\}) < c$. In fact, upon backtracking to any decision level b where $j \leqslant b < c$ the residual of D under the trail becomes unit. This holds in particular for $b = c - 1$. There exist subsequences P and Q of I such that $I = PQ$ and $\delta(P) = b$ and a literal $\ell \in D$ such that $D|_P = \{\ell\}$. We backtrack to decision level b using D as a reason. As for rule BackTrue, all literals with decision level greater than b are unassigned after which ℓ is propagated obtaining the trail $PK\ell$ where $\delta(PK\ell) = b$.

Decide. The trail I neither satisfies nor falsifies F and there are no unit literals in $F|_I$. An unassigned variable is chosen and assigned to decision level $d = \delta(I) + 1$.

Example 16.2. *We now explain our calculus by a small example, precisely, the one for which our calculus in [137] would fail if incorrectly blocking clauses and dual reasoning were combined.*

Let our formula be $F = (\overline{1} \vee 2) \wedge (1 \vee 2)$ over variables $V = \{1, 2, 3\}$ similar to DIMACS. The execution trace is shown in Figure 16.4. The procedure starts with the

1 In contrast to [138] we require ℓ to be the negation of the last decision which corresponds to learning the decision clause instead of the more common first UIP clause. Lifting this restriction is part of future work.

EndTrue:	$(F, I, M, \delta) \rightsquigarrow_{\mathsf{EndTrue}} M \vee I$ if $F\vert_I = \top$ and $\mathsf{decs}(I) = \emptyset$
EndFalse:	$(F, I, M, \delta) \rightsquigarrow_{\mathsf{EndFalse}} M$ if exists $C \in F$ and $C\vert_I = \bot$ and $\delta(C) = 0$
Unit:	$(F, I, M, \delta) \rightsquigarrow_{\mathsf{Unit}} (F, I\ell, M, \delta[\ell \mapsto a])$ if $F\vert_I \neq \top$ and $\bot \notin F\vert_I$ and exists $C \in F$ with $\{\ell\} = C\vert_I$ and $a = \delta(C \setminus \{\ell\})$
BackTrue:	$(F, I, M, \delta) \rightsquigarrow_{\mathsf{BackTrue}} (F, PK\ell, M \vee I, \delta[L \mapsto \infty][\ell \mapsto e])$ if $F\vert_I = \top$ and $PQ = I$ and $D = \overline{\mathsf{decs}(I)}$ and $e + 1 = \delta(D) = \delta(I)$ and $\ell \in D$ and $e = \delta(D \setminus \{\ell\}) = \delta(P)$ and $K = Q_{\leqslant e}$ and $L = Q_{>e}$
BackFalse:	$(F, I, M, \delta) \rightsquigarrow_{\mathsf{BackFalse}} (F, PK\ell, M, \delta[L \mapsto \infty][\ell \mapsto j])$ if exists $C \in F$ and exists D with $PQ = I$ and $C\vert_I = \bot$ and $c = \delta(C) = \delta(D) > 0$ such that $\ell \in D$ and $\overline{\ell} \in \mathsf{decs}(I)$ and $\ell\vert_Q = \bot$ and $F \wedge \overline{M} \models D$ and $j = \delta(D \setminus \{\ell\})$ and $b = \delta(P) = c - 1$ and $K = Q_{\leqslant b}$ and $L = Q_{>b}$
Decide:	$(F, I, M, \delta) \rightsquigarrow_{\mathsf{Decide}} (F, I\ell^d, M, \delta[\ell \mapsto d])$ if $F\vert_I \neq \top$ and $\bot \notin F\vert_I$ and $\mathsf{units}(F\vert_I) = \emptyset$ and $V(\ell) \in V$ and $\delta(\ell) = \infty$ and $d = \delta(I) + 1$

Figure 16.3: Rules for propositional model enumeration with chronological backtracking. The formula F takes variables in V. If I satisfies F and there is no decision at conflict level left, the search terminates (rule EndTrue). Otherwise the last decision literal is flipped (BackTrue). In case of a conflict at conflict level zero the search terminates (EndFalse). If I falsifies F at conflict level $c > 0$, a clause D is learned. The solver backtracks chronologically to decision level $c - 1$ where D becomes unit and propagates its unit literal (BackFalse). If the residual of F under I is undefined, either unit propagation is applied (Unit) or a decision taken (Decide).

Step	Rule	I	$\delta(1)$	$\delta(2)$	$\delta(3)$	$F\vert_I$	M
0		ε	∞	∞	∞	$(\bar{1} \vee 2) \wedge (1 \vee 2)$	$\bot$
1	Decide	3^d	∞	∞	1	$(\bar{1} \vee 2) \wedge (1 \vee 2)$	$\bot$
2	Decide	$3^d\,2^d$	∞	2	1	$\top$	$\bot$
3	BackTrue	$3^d\,\bar{2}$	∞	1	1	$(\bar{1}) \wedge (1)$	$3\,2$
4	Unit	$3^d\,\bar{2}\,\bar{1}$	1	1	1	$\bot$	$3\,2$
5	BackFalse	$\bar{3}$	∞	∞	0	$(\bar{1} \vee 2) \wedge (1 \vee 2)$	$3\,2$
6	Decide	$\bar{3}\,1^d$	1	∞	0	(2)	$3\,2$
7	Unit	$\bar{3}\,1^d 2$	1	1	0	$\top$	$3\,2$
8	BackTrue	$\bar{3}\,\bar{1}$	0	∞	0	(2)	$3\,2 \vee \bar{3}\,1\,2$
9	Unit	$\bar{3}\,\bar{1}\,2$	0	0	0	$\top$	$3\,2 \vee \bar{3}\,1\,2$
10	EndTrue						$3\,2 \vee \bar{3}\,1\,2 \vee \bar{3}\,\bar{1}\,2$

Figure 16.4: Execution trace for Example 16.2

empty trail and $M = \bot$ and by assigning all variables in V to decision level ∞ (step 0). Then, literal 3 is decided and assigned to decision level $d = 1$ (step 1). After literal 2 is decided and assigned to decision level $d = 2$, the model $3\,2$ is found (step 2). Now rule BackTrue *is executed with $D = (\bar{3} \vee \bar{2})$ and $e = 1$. We backtrack chronologically by unassigning literal 2 and then propagate its complement $\bar{2}$ by assigning it to decision level 1. The found model is recorded and $M = 3\,2$ (step 3). The unit literal $\bar{1}$ is propagated at decision level $a = 1$ due to the reason clause $(\bar{1} \vee 2)$ resulting in a conflict (step 4). The conflicting clause is $C = (1 \vee 2)$ and the conflict level is $c = 1$. Conflict analysis yields the clause $(\bar{3})$ and the jump level $j = 0$. We backtrack to decision level $b = 0$ by unassigning literals $\bar{1}$, $\bar{2}$ and 3 and propagate $\bar{3}$ by assigning it to decision level $j = 0$ (step 5), after which literal 1 is decided and assigned to decision level $d = 1$ (step 6). Due to the reason clause $C = (\bar{1} \vee 2)$ with decision level $a = 1$, literal 2 is propagated by assigning it to decision level 1 and model $\bar{3}\,1\,2$ is found (step 7). Rule* BackTrue *is executed with $D = (\bar{1})$ and $e = 0$. We backtrack chronologically by unassigning literals 2 and 1 and propagating literal $\bar{1}$ assigning it to decision level 0 (step 8). Literal 2 is propagated, the reason clause is $(1 \vee 2)$ at decision level $a = 0$, and model $\bar{3}\,\bar{1}\,2$ is found (step 9). There are no decisions left on I, and the model found in step 9 is added to M. The search terminates with $M = 3\,2 \vee \bar{3}\,1\,2 \vee \bar{3}\,\bar{1}\,2$ (step 10).*

The choice of the decision literal is non-deterministic and might have a significant impact on the trace length as we are going to show. Assume in step 1 we decide literal 2 assigning it decision level 1. We immediately find model 2 and apply rule BackTrue *with $D = (\bar{2})$ and $e = 0$, i.e., we flip the last decision literal 2 by unassigning it and assigning its complement to decision level 0. The residual of F under the trail is $(\bar{1}) \wedge (1)$, hence we apply rule* Unit *with $C = (\bar{1} \vee 2)$ and propagate literal $\bar{1}$ at decision level $a = 0$. We get a conflict with $C = (1 \vee 2)$ at decision level 0 and our procedure terminates with $M = 2$ after only four steps. If we start by deciding $\bar{2}$ instead, in the next step we might propagate literal $\bar{1}$ which leads to a conflict as in the last example. Rule* BackFalse *yields $D = (2)$ with $j = b = 0$ and after applying rule* Unit *and propagating literal 2 at decision level 0, we find model 2. Since the trail contains no decision literal, the procedure terminates by means of rule* EndTrue *in state $M = 2$ after four steps as well. Now let us decide*

$$
\begin{aligned}
&(1) \quad \forall k, \ell \in \mathsf{decs}(I)\,.\,\tau(I,k) < \tau(I,\ell) \implies \delta(k) < \delta(\ell)\\
&(2) \quad \delta(\mathsf{decs}(I)) = \{1,\ldots,\delta(I)\}\\
&(3) \quad \forall n \in \mathbb{N}\,.\,F \wedge \overline{M} \wedge \mathsf{decs}_{\leqslant n}(I) \models I_{\leqslant n}\\
&(4) \quad M \vee O(I) \text{ is a DSOP}\\
&(5) \quad M \vee F \wedge O(I) \equiv F
\end{aligned}
$$

Figure 16.5: Invariants for propositional model counting with conflict-driven clause learning and chronological backtracking.

literal 1 first. Then $I = 1^d$ and $F|_I = (2)$, literal 2 is propagated due to reason $(\bar{1} \vee 2)$ at decision level 1 and model $1\,2$ is found. By means of rule BackTrue *backtracking occurs to decision level 0 and the decision literal 1 is flipped. We now have $I = \bar{1}$ and $F|_I = (2)$. Again, literal 2 is propagated resulting in $I = \bar{1}\,2$ which satisfies F. The trail contains no decision literals, and rule* EndTrue *is applied. Hence, after five steps the procedure terminates in state $M = 1\,2 \vee \bar{1}\,2$. If the first decision is $\bar{1}$, analogously the following steps are executed: propagation of literal 2 finding the model $\bar{1}\,2$ (*Unit*), backtracking and flipping decision $\bar{1}$ (*BackTrue*), propagating literal 2 (*Unit*), finding model $1\,2$ and terminating with $M = \bar{1}\,2 \vee 1\,2$ (*EndTrue*). In total, five steps are required.*

We conclude this example by remarking that the trail is shortest if first literals are decided which occur either positively or negatively in all models. The worst case with respect to trail length is if literals are decided first whose variable does not appear in the formula at all. This effect is due to the nature of chronological backtracking.

16.5 PROOFS

For proving the correctness of our method, we make use of the invariants listed in Figure 16.5 the first two of which were introduced in [138], while the third one is Invariant (3) in [138] strengthened by the negation of M. This strengthening is required to show that after backtracking upon finding a model (rule BackTrue) the propagated literals are indeed implied. Their "reason" is the clause D consisting of the negated literals of the model just found. In contrast to rule BackFalse, D is not implied by the formula F but subsumed by the negation of a cube in its DSOP representation M instead, namely the model just found.

Invariants (4) and (5) represent statements concerning the pending search space needed to show that in the end state M is logically equivalent to the formula and thus their model counts coincide. Invariant (4) states that M is DSOP, hence, its model count equals the sum of the number of models of its cubes. Invariant (5) says that all models of F are represented by the formula obtained by the disjunction of M and the pending models of F. Thus, upon termination of the procedure the second disjunct is empty, i.e., M contains all models of F.

16.5.1 *Invariants in Non-Terminal States*

Lemma 16.1. *Invariants (1) – (5) hold in non-terminal states.*

The proof is carried out by induction over the number of rule applications. Assuming that Invariant (1) – (5) hold in a non-terminal state (F, I, M, δ), we show that they are met after the transition to another non-terminal state for all rules.

Unit

Invariants (1) and (2): Except for the DSOP M contained in the state, rule Unit matches the one in [138], hence Invariants (1) and (2) are met.

Invariant (3): The argument is the same as for Invariant (3) in [138] replacing F by $F \wedge \overline{M}$ where M remains unaltered, hence Invariant (3) also holds after the application of rule Unit.

Invariant (4): We have that $M \vee O(I)$ is a DSOP and we need to show that $M \vee O(I\ell)$ is a DSOP as well. According to its definition, $O(I\ell)$ is a DSOP. We have $O(I\ell) = I\ell \vee R(I\ell) = I\ell \vee R_{\leqslant a}(I\ell) \vee R_{>a}(I\ell)$. Since $\delta(\ell) = a$ and $\ell \notin \mathsf{decs}(I\ell)$,i ℓ does not occur in $R_{\leqslant a}(I\ell)$, hence $O(I\ell) = I\ell \vee R_{\leqslant a}(I) \vee R_{>a}(I) \wedge \ell$. Since $M \vee O(I)$ is a DSOP, $M \vee O(I\ell)$ is a DSOP as well.

Invariant (5): Since $F \wedge I \models \ell$, we have $F \wedge O(I) \equiv F \wedge O(I\ell)$. From this we get $M \vee F \wedge O(I) \equiv M \vee F \wedge O(I\ell) \equiv F$, and Invariant (5) holds after executing Unit.

BackTrue

Invariants (1) and (2): We need to prove that the order of the decisions left on the trail remains unaltered and no new decisions are taken. We show that (A) K contains no decision literal and (B) in the post state ℓ is not a decision literal either.

(A) We have $I = PQ$ and $K = Q_{\leqslant e}$, i.e., K is obtained from Q by removing all literals with decision level greater than e. Furthermore, for all $k \in K$, $p \in P$, we have $\tau(PK, p) < \tau(PK, k)$ and $\delta(K) \leqslant \delta(P) = e$. By the definition of decision literal and since Invariant (1) holds before applying BackTrue, the decision literal with decision level e is contained in P. Since K contains no literal with decision level greater than e, K contains no decision literal.

(B) It is sufficient to consider the case where $\delta(I) > 0$. We have $\delta(D \setminus \{\ell\}) = e = \delta(P)$ and $\delta(D) = e + 1$. According to Invariants (1) and (2) there exists exactly one decision literal for each decision level, and since $D = \overline{\mathsf{decs}(I)}$ and $\ell \in D$, $\overline{\ell} \in \mathsf{decs}(I)$. Furthermore, $\overline{\ell} \in Q$, hence $\overline{\ell}$ is unassigned upon backtracking, i.e., $\{\ell, \overline{\ell}\} \notin PK$, and $D|_{PK} = \{\ell\}$. Due to the definition of D there exists a literal $k \in D$ where $k \neq \ell$ such that $\delta(k) = \delta(\ell) = e$ for which $\tau(PK, k) < \tau(PK, \ell)$. Due to Invariant (1) and the definition of decision literal, ℓ is not a decision literal. The order of the decision literals left on the trail is not affected, hence Invariant (1) holds. Also, the remaining decisions remain unaltered, and Invariant (2) holds as well.

Invariant (3): We need to show $F \wedge \overline{(M \vee I)} \wedge \mathsf{decs}_{\leqslant n}(PK\ell) \models (PK\ell)_{\leqslant n}$ for all n. First note that the decision levels of all the literals in PK do not change while applying the rule. Only the decision level of the variable of ℓ is decremented from $e + 1$ to e. It also stops being a decision variable. Since $\delta(PK\ell) = e$, we can assume $n \leqslant e$. Observe $F \wedge \overline{(M \vee I)} \wedge \mathsf{decs}_{\leqslant n}(PK\ell) \equiv \bar{I} \wedge (F \wedge \overline{M} \wedge \mathsf{decs}_{\leqslant n}(I))$ since ℓ is not a decision in $PK\ell$ and $I_{\leqslant e} = PK$ and thus $I_{\leqslant n} = (PK)_{\leqslant n}$ by definition. Now the induction hypothesis is applied and we get $F \wedge \overline{(M \vee I)} \wedge \mathsf{decs}_{\leqslant n}(PK\ell) \models I_{\leqslant n}$. Again using $I_{\leqslant n} = (PK)_{\leqslant n}$ this almost closes the proof except that we are left to prove $F \wedge \overline{(M \vee I)} \wedge \mathsf{decs}_{\leqslant e}(PK\ell) \models \ell$ as ℓ has decision level e in $PK\ell$ after applying the rule and thus ℓ disappears in the proof obligation for $n < e$. To see this note that $F \wedge \overline{D} \models I$ using again the induction hypothesis for $n = e + 1$. This gives $F \wedge \overline{\mathsf{decs}_{\leqslant e}(PK)} \wedge \overline{\ell} \models I$ and thus $F \wedge \overline{\mathsf{decs}_{\leqslant e}(PK)} \wedge \bar{I} \models \ell$ by conditional contraposition.

Invariant (4): We assume $M \vee O(I)$ is a DSOP and need to show that $(M \vee I) \vee O(PK\ell)$ is a DSOP as well. Now $O(I) = I \vee R_{\leqslant e+1}(I)$ since $\delta(I) = e+1$. Further, $O(I) = I \vee R_{\leqslant e}(I) \vee R_{=e+1}(I)$. The pending search space of $PK\ell$ is equal to $O(PK\ell) = PK\ell \vee R_{\leqslant e}(PK\ell)$. But $PK = I_{\leqslant e}$ and $PK\ell = I_{\leqslant e}\ell = R_{=e+1}(I)$ since $\overline{\ell} \in \mathsf{decs}(I)$ and $\delta(\overline{\ell}) = e+1$. In addition, $R_{\leqslant e}(PK\ell) = R_{\leqslant e}(PK)$ since $\ell \notin \mathsf{decs}(PK\ell)$ and $\delta(\ell) = e$ which gives us $R_{\leqslant e}(PK\ell) = R_{\leqslant e}(I)$. We have $O(PK\ell) = R_{=e+1}(I) \vee R_{\leqslant e}(I)$. From this we get $O(PK\ell) \vee I = O(I)$ and $(M \vee I) \vee O(PK\ell) = M \vee (I \vee O(PK\ell)) = M \vee O(I)$ which is a DSOP, and Invariant (4) holds.

Invariant (5): Given $M \vee (F \wedge O(I)) \equiv F$, we need to show that $(M \vee I) \vee (F \wedge O(PK\ell)) \equiv F$. From $O(PK\ell) \vee I = O(I)$ we get $M \vee F \wedge O(I) = M \vee F \wedge (O(PK\ell) \vee I) = M \vee (F \wedge O(PK\ell)) \vee (F \wedge I)$. But $I \models F$ and $F \wedge I \equiv I$. Therefore $M \vee F \wedge O(I) = M \vee (F \wedge O(PK\ell)) \vee I = (M \vee I) \vee F \wedge O(PK\ell) \equiv F$, and Invariant (5) is met.

BackFalse

Invariants (1) and (2): Except for the DSOP M added to the state, the fact that clause D is not added to F and that $F \wedge \overline{M} \models D$, rule BackFalse corresponds to rule Jump in [138] where $b = c - 1$. Therefore, rule BackFalse is a special case of rule Jump and Invariants (1) – (2) still hold after the execution of rule BackFalse.

Invariant (3): Let n be arbitrarily fixed. Before executing BackFalse, $F \wedge \overline{M} \wedge \mathsf{decs}_{\leqslant n}(I) \models I_{\leqslant n}$. We need to show that $F \wedge \overline{M} \wedge \mathsf{decs}_{\leqslant n}(PK\ell) \models (PK\ell)_{\leqslant n}$. We have $I = PQ$ and $PK < I$, i.e., $F \wedge \overline{M} \wedge \mathsf{decs}_{\leqslant n}(PK) \models (PK)_{\leqslant n}$. From $j = \delta(D \setminus \{\ell\})$, $c = \delta(D)$ and $\delta(P) = c - 1$ we get $D|_{PK} = \{\ell\}$. On the one hand $F \wedge \overline{M} \wedge \mathsf{decs}_{\leqslant n}(PK) \models \overline{D \setminus \{\ell\}}$ and on the other hand $F \wedge \overline{M} \wedge \mathsf{decs}_{\leqslant n}(PK) \models D$, therefore, by modus ponens, $F \wedge \overline{M} \wedge \mathsf{decs}_{\leqslant n}(PK) \models \ell$. Since ℓ is not a decision literal, as shown above, $F \wedge \overline{M} \wedge \mathsf{decs}_{\leqslant n}(PK) \equiv F \wedge \overline{M} \wedge \mathsf{decs}_{\leqslant n}(PK\ell)$ and $F \wedge \overline{M} \wedge \mathsf{decs}_{\leqslant n}(PK\ell) \models (PK\ell)_{\leqslant n}$. So, Invariant (3) holds after applying rule BackFalse.

Invariant (4): Given that $M \vee O(I)$ is a DSOP, we need to show that $M \vee O(PK\ell)$ is a DSOP as well. As shown for rule BackTrue, $O(PK\ell) \vee I = O(I)$ if $\overline{\ell} \in \mathsf{decs}(I)$. Due to the premise, we have $M \wedge O(I) \equiv \bot$. Therefore, $M \wedge O(I) = M \wedge (O(PK\ell) \vee I) \equiv (M \wedge O(PK\ell)) \vee (M \wedge I) \equiv \bot$, in particular $M \wedge O(PK\ell) \equiv \bot$, and Invariant (4) holds.

Invariant (5): Given that $M \vee (F \wedge O(I)) \equiv F$, we need to show that $M \vee (F \wedge O(PK\ell)) \equiv F$ as well. Analogously to rule BackTrue we have $M \vee F \wedge O(I) = M \vee (F \wedge O(PK\ell)) \vee (F \wedge I)$. But $F \wedge I \equiv \bot$, hence $M \vee F \wedge O(I) = M \vee F \wedge O(PK\ell) \equiv F$, and Invariant (5) is met.

Decide

Invariants (1) and (2): Except for the DSOP M contained in the state, rule Decide matches exactly the one in [138], and, following the argument given there, Invariants (1) – (2) are met.

Invariant (3): Let n be arbitrarily fixed. Note that literal ℓ is a decision literal. Therefore $F \wedge \overline{M} \wedge \mathsf{decs}_{\leqslant n}(I\ell) \equiv F \wedge \overline{M} \wedge \mathsf{decs}_{\leqslant n}(I) \wedge \ell \models I_{\leqslant n}(I) \wedge \ell \equiv (I\ell)_{\leqslant n}$, and Invariant (3) holds.

Invariant (4): Assuming $M \vee O(I)$ is a DSOP, we need to show that $M \vee O(I\ell)$ with decision literal ℓ is a DSOP as well. We have $O(I\ell) = O_{\leqslant d}(I\ell)$ where $d = \delta(I) + 1$ since $\ell \in \mathsf{decs}(I\ell)$ and $\delta(\ell) = d$. Further, $O(I\ell) = I\ell \vee R_{\leqslant d}(I\ell) = I\ell \vee$

$R_{\leqslant d-1}(I\ell) \vee R_{=d}(I\ell) = I\ell \vee R_{\leqslant d-1}(I\ell) \vee \overline{\ell} \wedge I_{\leqslant d-1}$ where $\delta(I) = d-1$. Observing that $\ell \notin (I\ell)_{=i}$ for $i < d$, i.e., $(I\ell)_{\leqslant i} = I_{\leqslant i}$ for $i \leqslant d-1 = \delta(I)$, we get $O(I\ell) = I\ell \vee R(I) \vee \overline{\ell} I = I(\ell \vee \overline{\ell}) \vee R(I) = I \vee R(I) = O(I)$. This gives us $M \vee O(I\ell) = M \vee O(I)$ which due to our premise is a DSOP, and Invariant (4) holds.

Invariant (5): As just shown, $O(I\ell) \equiv O(I)$. Therefore, $M \vee F \wedge O(I\ell) \equiv M \vee F \wedge O(I) \equiv F$, and after executing rule Decide Invariant (5) still holds.

16.5.2 *Equivalence and Model Count*

Proposition 16.1. *If* Enumerate *reaches a terminal state M, then $M \equiv F$ and the model count of F is given by $\#F = \sum_{C \in M} 2^{|V-C|}$.*

Due to Invariants (4) and (5), in every non-terminal state M is a DSOP of models of F, but we still need to show that the resulting $I \vee M$ in EndTrue and M in EndFalse are equivalent to F.

EndTrue. Prior to applying rule EndTrue, $M \vee O(I)$ is a DSOP and $M \vee F \wedge O(I) \equiv F$. Since $\mathsf{decs}(I) = \emptyset$, we have $R(I) = \bot$ and $O(I) = I$, i.e., $M \vee I$ is a DSOP as well. Since $I \models F$, we have $F \wedge I \equiv I$ and $M \vee (F \wedge O(I)) \equiv M \vee (F \wedge I) \equiv M \vee I \equiv F$ due to the premise.

EndFalse. For showing that $M \vee F \wedge O(I) \equiv F$ we observe that $R(I) = R_{\leqslant \delta(C)}(I) = \bot$. We therefore have $O(I) = I \vee R(I) = I \vee \bot = I$. Furthermore, I falsifies F, i.e., $F \wedge I \equiv \bot$. This gives us $M \vee F \wedge O(I) \equiv M \vee F \wedge I \equiv M \equiv F$ due to the premise.

16.5.3 *Progress*

Proposition 16.2. Enumerate *always makes progress, i.e., in every non-terminal state a rule is applicable.*

The proof is conducted by induction over the number of rule applications. We show that in any non-terminal state (F, I, M, δ) a rule can be executed.

Assume $F|_I = \top$. If $\mathsf{decs}(I) = \emptyset$, rule EndTrue can be applied. Otherwise we choose $D = \overline{\mathsf{decs}(I)}$. We have $\delta(D) = \delta(I) = e+1$ and due to Invariant (2) D contains exactly one decision literal ℓ such that $\delta(\ell) = e+1$ and therefore $\delta(D) \setminus \{\ell\}) = e$. We choose P and Q such that $I = PQ$ and $e = \delta(P)$ and in particular $\ell|_Q = \bot$. After backtracking to decision level e, we have $I_{\leqslant e} = PK$ where $K = Q_{\leqslant e}$ and $D|_{PK} = \{\ell\}$. All preconditions of rule BackTrue are met.

If instead $F|_I = \bot$, there exists a clause $C \in F$ such that $C|_I = \bot$. If $\delta(C) = 0$, rule EndFalse is applicable. Otherwise, by Invariant (3) we have $F \wedge \overline{M} \wedge \mathsf{decs}_{\leqslant c}(I) \equiv F \wedge \overline{M} \wedge \mathsf{decs}_{\leqslant c}(I) \wedge I_{\leqslant c} \models I_{\leqslant c}$. Since $I_{\leqslant c}(F) \equiv \bot$, also $F \wedge \overline{M} \wedge \mathsf{decs}_{\leqslant c}(I) \wedge I_{\leqslant c} \equiv F \wedge \overline{M} \wedge \mathsf{decs}_{\leqslant c}(I) \equiv \bot$. We choose $\overline{D} = \mathsf{decs}(I)$ obtaining $F \wedge \overline{M} \wedge \overline{D} \wedge I_{\leqslant c} \equiv \bot$, thus $F \wedge \overline{M} \models D$. Similarly to the proof of progress given in Prop. 2 in [138], but for $b = c-1$, it can be shown that all preconditions of rule BackFalse hold. Note that we do not explicitly add D to F since in CDCL with chronological backtracking we need no blocking clauses. We use D exclusively to determine ℓ instead.

The proof for the case where $F|_I \notin \{\top, \bot\}$ is identical to the one for Prop. 2 in [138], since apart from M in non-terminal states rules Unit and Decide are equal in both frameworks.

Since all possible cases are covered by this argument, in every non-terminal state a rule is applicable, i.e., Enumerate always makes progress.

16.5.4 *Termination*

Proposition 16.3. ENUMERATE *always terminates, i.e., no infinite state sequence is generated.*

The proof is analogous to the one in [138] with rule BackFalse replacing Jump and the additional rule BackTrue to which the observations concerning BackFalse apply as well.

16.6 CONCLUSION

The results for combining chronological backtracking with CDCL presented in [149] and its potential for propositional model counting conjectured in [138] provided the main motivation for our work. We have presented a formal calculus for propositional model counting based on these ideas and provided a formal proof of its correctness.

For our framework we chose a model enumeration approach. The main motivation therefore was the following. Let F be a formula and I the current trail. A statement of an invariant taking into account the model count alone would be something like $MC + MO = MF$ where MC denotes the number of total models of F found so far, MO denotes the number of pending total models of F and MF denotes the model count of F. But only one of the three quantities is known, namely MC, whereas the newly introduced Invariants (4) and (5) allow first for a precise characterization of both the found and pending models and second to show that the union of the two yields the models of F.

The preconditions $F|_I \neq \top$ and $\bot \notin F|_I$ may be omitted in rules Unit and Decide without compromising the procedure's correctness. The most important remaining open question is whether literal ℓ in rule BackFalse needs to be a flipped decision.

We plan to implement our rules to experimentally validate their effectiveness and to investigate possible applications in SMT and QBF. Our hope is to avoid the overhead of introducing blocking clauses in order to make use of learning. We want to extend the presented approach to projected model counting, also in combination with dual reasoning [137]. We further target component-based reasoning.

17

DISCUSSION OF PAPER 4

The main contributions are pointed out, the choice of an enumeration approach is motivated, and proof details not stated explicitly in the paper are given (Section 17.1). The concept of pending search space is central in our framework but does not occur explicitly in our rules. In Section 17.2, we show its evolution during the execution of our calculus by means of an example and identify a weak spot in its definition and in our rules: its definition in the initial state is not clear, and in the paper, we state that, for a formula F and a trail I over variables of F, upon termination it holds that $F \wedge O(I)$ is empty. However, none of $\mathsf{EndTrue}$ and $\mathsf{EndFalse}$ alter I, hence $F \wedge O(I) = I$ in the former case, and our statement does not hold if the computation terminates with a model. The proof of our calculus remains unaffected. Nevertheless, we believe this should be fixed, and we are going to present preliminary ideas in Section 17.3.

17.1 MAIN CONTRIBUTIONS

Our goal was to devise a model counting approach based on chronological CDCL by extending our former calculus [138] accordingly and to provide a formal proof of its correctness. In place of the model count, our method computes a DSOP representation of the input formula. However, it can readily be adapted to compute the model count of the input formula by summing up the number of total assignments represented by the detected (partial) models.

The enumeration approach facilitates the proof of correctness of our method: for the formula F, a trail I over the variables of F, and a DSOP formula M consisting of (partial) models of F, it holds anytime that $M \vee (F \wedge O(I)) \equiv F$, and therefore also $\#(F \wedge O(I)) = \#F$. The new concept of pending search space, denoted by $O(I)$, describes the assignments not yet tested. The models still to be found are given by $F \wedge O(I)$.

Our proof extends the proof of our former work Chapter 14. We introduce new invariants involving the pending search space and the pending models. Invariant (4) is crucial, since it is the one expressing that every total assignment is tested at most once. In fact, every total assignment is tested *exactly* once, albeit its check might occur implicitly, namely if it is a total extension of either a partial model or a partial counter-model of F. In this case, backtracking occurs without assigning all variables. The strengthening of Invariant (3) by $\neg M$ is similar to the case where blocking clauses are added to F: chronological backtracking ensures that every total assignment is tested exactly once, and therefore the found models (and counter-models) are "blocked" in the sense that they can not be found again. Consequently, propagated literals are implied by $F \wedge \neg M \wedge \mathsf{decs}_{\leqslant n}(I)$.

Table 17.1: Execution trace for model counting using chronological CDCL.

S	RULE	I	$F\|_I$	$O(I)$	M	CM
0		ε	F	1	0	0
1	Decide	a^d	$(\neg b)$	$(\neg a) \vee (a)$	0	0
2	Unit	$a^d \neg b^{C_3}$	1	$(\neg a) \vee (a \wedge \neg b)$	0	$(a \wedge b)$
3	BackTrue	$\neg a$	$(\neg b) \wedge (\neg c)$	$(\neg a)$	$(a \wedge \neg b)$	$(a \wedge b)$
4	Unit	$\neg a \neg b^{C_1}$	$(\neg c)$	$(\neg a \wedge \neg b)$	$(a \wedge \neg b)$	$(a \wedge b) \vee (\neg a \wedge b)$
5	Unit	$\neg a \neg b^{C_1} \neg c^{C_2}$	1	$(\neg a \wedge \neg b \wedge \neg c)$	$(a \wedge \neg b)$	$(a \wedge b) \vee (\neg a \wedge b) \vee (\neg a \wedge \neg b \wedge c)$
6	EndTrue	$\neg a \neg b^{C_1} \neg c^{C_2}$	1	$(\neg a \wedge \neg b \wedge \neg c)$	$(a \wedge \neg b) \vee (\neg a \wedge \neg b \wedge \neg c)$	$(a \wedge b) \vee (\neg a \wedge b) \vee (\neg a \wedge \neg b \wedge c)$

17.2 THE PENDING SEARCH SPACE BY AN EXAMPLE

Consider the propositional formula

$$F = \underbrace{(a \vee \neg b)}_{C_1} \wedge \underbrace{(a \vee \neg c)}_{C_2} \wedge \underbrace{(\neg a \vee \neg b)}_{C_3}$$

defined over the set of variables $V = \{a, b, c\}$. It has three models: $a \neg b c$, $a \neg b \neg c$, and $\neg a \neg b \neg c$. The calculus depicted in Figure 16.3 generates the execution trace listed in Table 17.1. The second column denotes the rule which is applied. The third and fourth column denote the resulting trail I and the residual of F under I. The fifth and sixth column refer to the pending search space with respect to I and the DSOP formula M consisting of the models of F, while CM is a DSOP containing the counter-models of F as will be explained further down.

Step 0: Initially, the trail I is empty, and $O(I)$ consists of the disjunction of all possible assignments to the variables in V, which is equivalent to 1.

Step 1: The decision a^d is taken, and $I = a^d$. The pending search space is defined as $O(a^d) = (a) \vee (\neg a)$, i. e., it consists of I and the right branch of the decision a^d, as defined in Section 16.3. Notably, $O(a^d) \equiv 1$, i. e., the pending search space remains the same: a decision splits the pending search space into two halves, namely one in which the decision literal is assigned the value 1 and one in which it is assigned 0.

Step 2: The literal $\neg b$ is propagated with reason C_3, and $I = a^d \neg b^{C_3}$ is a model of F. The pending search space is given by $O(I) = (\neg a) \vee (a \wedge \neg b)$, which is not logically equivalent to the pending search space in the former step. In fact, by propagating $\neg b$, we rule out the assignment ab, which is a counter-model of F and is shown in the last column entitled CM for "counter-models", in which we keep track of the assignments ruled out by executing unit propagation.

Step 3: Backtracking occurs, and the most recent decision literal is flipped. The resulting trail is $I = \neg a$. It contains no decision literal, and hence $O(I) = (\neg a)$, which is exactly the pending search space of the previous step from which the model $m_1 = a\neg b$ just found has been removed.

Step 4: The literal $\neg b$ is propagated with reason C_1, and accordingly, the assignment $\neg a\, b$ is removed from the pending search space of the previous step and added to MC, since it can not be extended to a model of F.

Step 5: Again, the Unit rule is executed, and the corresponding assignment $\neg a\neg b c$ is added to MC. The trail $I = \neg a\neg b^{C_1}\neg c^{C_2}$ contains no decision, and the pending search space is given by $O(I) = (\neg a \wedge \neg b \wedge \neg c)$.

Step 6: The computation terminates with the execution of rule EndTrue in state M. The trail remains unaltered, and consequently the pending search space is not empty, either, although the search space has been processed exhaustively.

17.3 TOWARDS AN ALTERNATIVE TERMINATION CONDITION

The pending search space consists of the assignments which are not yet tested. It provides a means to describe the pending models without knowing them, as in Invariant (5), and is an important ingredient in our proof. However, the example in Section 17.2 revealed two issues concerning the pending search space.

First, by the definition introduced in Section 16.3, the pending search space of an empty trail is given by $O(\varepsilon) = 0$. However, while in the initial state the trail is empty, the whole search space still need be processed and the pending search space should therefore be defined as $O(I) = 1$ in the initial state.

Second, one would expect the pending search space to be empty after termination, since the search space has been processed exhaustively. This is actually not the case in our calculus, since neither rule EndTrue nor rule EndFalse alters I. The computation therefore terminates with $O(I) \neq 0$, although all assignments have been tested, and even with $F \wedge O(I) \neq 0$ if I is a model of F, although all models have been found and recorded. This is rather counter-intuitive. This issue can be fixed by setting $I = \varepsilon$ in rules EndTrue and EndFalse. This is correct: Let J denote a trail. To meet the preconditions for a termination rule, the trail J contains only propagated literals and is therefore the last assignment which need be tested. Accordingly, the pending search space is $O(J) = J$, and after executing a terminal rule, the assignment represented by J need be "subtracted" from both the trail J and the pending search space $O(J)$. Obviously, both become empty.

In our framework, the computation terminates as soon as a terminal state is reached, because no further rule is applicable. However, if the termination rules alter the trail, it need be contained in the terminal state, too, which could be defined as (M, I) or even (M, ε). This solution still ensures that no rule is applicable to a terminal state, and no additional termination condition is required.

Things look differently if we want to represent both intermediate and terminal states similarly, for instance according to the definition in our paper: Let an intermediate state be $s_i = (F, M, I, \delta)$, where I contains only propagated literals and is a model of F. If we alter rule EndFalse as described above, after its execution we reach the terminal state $s_f = (F, M, \varepsilon, \delta_0)$. But if $M = 0$, we can not distinguish s_f from the initial state $s_0 = (F, 0, \varepsilon, \delta_0)$, and the computation might start over again. The problem consists in the definition of the trail: it is empty in both the initial and the terminal states, and the decision level function is equal in those states, too.

However, the pending search state differs in the initial and terminal states. We could therefore define intermediate states as $s_i = (F, I, M, \delta, O)$, where $O = O(I)$.

The initial state is then given by $s_0 = (F, \varepsilon, 0, \delta_0, 1)$. The terminal state would be defined as $s_f = (F, \varepsilon, M, \delta_0, 0)$, and the termination condition would be $O = 0$.

Adding one element to the state representation for providing a termination criterion might seem ugly. However, none of the other elements can be discarded. Furthermore, considering the pending search space but not the processed search space or the counter-models introduces some kind of asymmetry. This can be remediated by extending the state representation further. First, observe that the processed search space in a state (F, I, M, δ, O) is given by the disjunction of M and CM, which denotes the counter-models found during computation, similarly to the example in Section 17.2. Indeed, these are the only assignments which are not explicitly checked, and therefore the processed search space can be described as $M \vee CM$. Intermediate states could then be defined as $s_i = (F, I, M, \delta, O, C)$ with the initial and terminal states defined accordingly.

In both definitions, some redundancy is introduced by the addition of the pending search space $O(I)$, since it can be computed from the trail I in every but the initial state. Its usefulness is therefore rather limited. An option could be to replace it by a flag expressing whether we are in a terminal (or initial) state or not. This addition is useful for the proofs but not necessary in an implementation.

In this part, we worked on combining DPLL for #SAT with backjumping but we did not focus on finding short models, which improves the search behavior (Part iii). In the next part, we focus on approaches to shorten models beyond the dual approach.

Part V

PARTIAL MODEL ENUMERATION

18

PAPER 5: FOUR FLAVORS OF ENTAILMENT

PUBLISHED. In: *Theory and Applications of Satisfiability Testing – SAT 2020 – 23rd International Conference, Alghero, Italy, July 3–10, 2020, Proceedings*. Ed. by Luca Pulina and Martina Seidl. Vol. 12178. Lecture Notes in Computer Science. Springer, 2020, pp. 62–71. DOI: 10.1007/978-3-030-51825-7_5.

AUTHORS. Sibylle Möhle, Roberto Sebastiani and Armin Biere.

ACKNOWLEDGMENTS. The work was supported by the LIT Secure and Correct Systems Lab funded by the State of Upper Austria, by the QuaSI project funded by D-Wave Systems Inc., and by the Italian Assocation for Artificial Intelligence (AI*IA). We thank the anonymous reviewers and Mathias Fleury for suggesting many textual improvements.

ABSTRACT. We present a novel approach for enumerating partial models of a propositional formula, inspired by how theory solvers and the SAT solver interact in lazy SMT. Using various forms of dual reasoning allows our CDCL-based algorithm to enumerate partial models with no need for exploring and shrinking full models. Our focus is on model enumeration without repetition, with potential applications in weighted model counting and weighted model integration for probabilistic inference over Boolean and hybrid domains. Chronological backtracking renders the use of blocking clauses obsolete. We provide a formalization and examples. We further discuss important design choices for a future implementation related to the strength of dual reasoning, including unit propagation, using SAT or QBF oracles.

18.1 INTRODUCTION

Model enumeration is a key task in various activities, such as lazy Satisfiability Modulo Theories [176], predicate abstraction [118], software product line engineering [80], model checking [21, 131, 184], and preimage computation [119, 180].

Whereas in some applications enumerating models multiple times causes no harm, in others avoiding repetitions is crucial. Examples are weighted model counting (WMC) for probabilistic reasoning in Boolean domains and weighted model integration (WMI), which generalizes WMC for hybrid domains [142, 143]. There, the addends are *partial* satisfying assignments, i.e., some variables remain unassigned. Each of these assignments represents a set of *total* assignments, and consequently, the number of the addends is reduced. A formula might be represented in a concise manner by the disjunction of its pairwise contradicting partial models, which is of interest in digital circuit synthesis [17]. Partial models are relevant also in predicate abstraction [118], preimage computation [119, 180], and existential quantification [32]. They can be obtained by shrinking total models [190]. Alternatively, dual reasoning, where the formula is considered together with its negation, allows for pruning the search space early and detecting partial models. It is also applied in the context of model counting [24, 137].

If only a subset X of the variables is significant, the models are *projected* onto these *relevant* variables. We say that we *existentially quantify* the formula over the *irrelevant* variables Y and write $\exists Y\,[\,F(X,Y)\,]$, where $F(X,Y)$ is a formula over variables X and Y such that $X \cap Y = \emptyset$. Projected model enumeration occurs in automotive configuration [203], existential quantifier elimination [32], image computation [94, 95], predicate abstraction [118], and bounded model checking [184].

To avoid finding models multiple times, blocking clauses might be added to the formula under consideration [104, 131]. This method suffers from a potentially exponential blowup of the formula and consequent slowdown of unit propagation. Toda and Soh [191] address this issue by a variant of conflict analysis, which is motivated by Gebser et al. [83] and is exempt from blocking clauses. Chronological backtracking in Grumberg et al. [94] and our previous work [139] ensures that the search space is traversed in a systematic manner, similarly to DPLL [60], and the use of blocking clauses is avoided. Whenever a model is found, the last (relevant) decision literal is flipped. No clause asserting this flipped decision is added, which might cause problems during later conflict analysis. This problem is addressed by modifying the implication graph [94] or by an alternative first UIP scheme [191].

Our contribution. We lift the way how theory and SAT solver interact in SMT to propositional projected model enumeration without repetition. Based on the notion of logical entailment, combined with dual reasoning, our algorithm detects partial models in a forward manner, rendering model shrinking superfluous. The test for entailment is crucial in our algorithm. Anticipating a future implementation, we present it in four flavors with different strengths together with examples. The main enumeration engine uses chronological CDCL [149], is exempt from blocking clauses, and thus does not suffer from a formula blowup. Its projection capabilities make it suitable also for applications requiring model enumeration with projection. We conclude our presentation by a formalization of our algorithm and a discussion of the presented approach. Our work is motivated by projected model counting and weighted model integration. We therefore focus on (projected) model enumeration without repetition. Contrarily to Oztok and Darwiche [161], we use an oracle and build a Disjoint Sum-of-Products (DSOP) [17]. The work by Lagniez and Marquis [116] is orthogonal to ours. It is led by a disjunctive decomposition of the formula under consideration after a full model is found and also decomposes it into disjoint connected components.

18.2 PRELIMINARIES

A *literal* ℓ is a variable v or its negation $\neg v$. We denote by $V(\ell)$ the variable of ℓ and extend this notation to sets and sequences of literals. We write $\overline{\ell}$ for the complement of ℓ, i.e., $\overline{\ell} = \neg\ell$, defining $\neg\neg\ell = \ell$. A formula in *conjunctive normal form (CNF)* over variables V is defined as a conjunction of *clauses*, which are disjunctions of literals with variable in V, whereas a formula in *disjunctive normal form (DNF)* is a disjunction of *cubes*, which are conjunctions of literals. We might interpret formulae, clauses, and cubes also as sets of clauses or cubes, and literals and write $C \in F$ for referring to a clause or cube C in a formula F and $\ell \in C$ where ℓ is a literal in C. The empty CNF formula and the empty cube are denoted by 1, the empty DNF formula and the empty clause by 0.

A *total assignment* is a mapping from the set of variables V to the truth values 1 (true) and 0 (false). A *trail* $I = \ell_1 \dots \ell_n$ is a non-contradictory sequence of literals, which might also be interpreted as a *(possibly partial) assignment*, where $I(\ell) = 1$ if $\ell \in I$ and $I(\ell) = 0$ if $\neg\ell \in I$. We denote the empty trail by ε and the set of variables of the literals on I by $V(I)$. Trails and literals might be concatenated, written $I = JK$ and $I = J\ell$, provided $V(J) \cap V(K) = \emptyset$ and $V(J) \cap V(\ell) = \emptyset$. We interpret I also as a set of literals and write $\ell \in I$ to denote a literal ℓ on I. The *residual* of a formula F under a trail I, written $F|_I$, is obtained by replacing the literals ℓ in F, where $V(\ell) \in V(I)$, by their truth value, and by recursively propagating truth values through Boolean connectives. In particular, for a CNF formula this consists in removing satisfied clauses as well as falsified literals. By "=" in $F|_I = 1$ and $F|_I = 0$, notably by omitting quantifiers, we explicitly mean syntactical equality and consider the (possibly partial) assignment represented by I, i.e., only the literals on I. The notion of residual is extended similarly to clauses and literals. We denote by $X - I$ the unassigned variables in X. By $\pi(I, X)$ we refer to the projection of I onto X and extend this notation to sets of literals.

The *decision level function* $\delta\colon V \mapsto \mathbb{N} \cup \{\infty\}$ returns the decision level of a variable v. If v is unassigned, we have $\delta(v) = \infty$, and δ is updated whenever v is assigned or unassigned. We define $\delta(\ell) = \delta(V(\ell))$ for a literal ℓ, $\delta(C) = \max\{\delta(\ell) \mid \ell \in C\}$ for a clause $C \neq 0$, and $\delta(I) = \max\{\delta(\ell) \mid \ell \in I\}$ for a sequence of literals $I \neq \varepsilon$. Further, $\delta(L) = \max\{\delta(\ell) \mid \ell \in L\}$ for a set of literals $L \neq \emptyset$. We define $\delta(0) = \delta(\varepsilon) = \delta(\emptyset) = 0$. The updated function δ, in which $V(\ell)$ is assigned to decision level d, is denoted by $\delta[\ell \mapsto d]$. If all literals in V are unassigned, we write $\delta[V \mapsto \infty]$ or $\delta \equiv \infty$. The function δ is left-associative, i.e., $\delta[I \mapsto \infty][\ell \mapsto d]$ first unassigns all literals on I and then assigns literal ℓ to decision level d. We mark the decision literals on I by a superscript, i.e., ℓ^d, and denote the set consisting of the decision literals on I by $\mathsf{decs}(I) = \{\ell \mid \ell^d \in I\}$. Similarly, we denote the set of unit literals in F or its residual under I by $\mathsf{units}(F)$ or $\mathsf{units}(F|_I)$. Trails are partitioned into *decision levels*, and $I_{\leqslant n}$ is the subsequence of I consisting of all literals ℓ where $\delta(\ell) \leqslant n$.

Following Sebastiani [177], we say that a (partial) assignment I *entails* a formula F, if all total extensions of I satisfy F. In this work it was noticed that, if I entails F, we can not conclude that $F|_I = 1$, but only that $F|_I$ is valid. Consider as an example $F = (x \wedge y) \vee (x \wedge \neg y)$ over variables $X = \{x\}$ and $Y = \{y\}$ and the trail $I = x$ ranging over $X \cup Y$. The possible extensions of I are $I' = xy$ and $I'' = x\neg y$. We have $F|_{I'} = F|_{I''} = 1$, therefore I entails F. Notice that $F|_I = y \vee \neg y$ is valid but it syntactically differs from 1.

18.3 EARLY PRUNING FOR PROJECTED MODEL ENUMERATION

Our approach is inspired by how theory solvers and the SAT solver interact in lazy SMT. A general schema is described in Figure 18.1. Let $F(X, Y)$ be a formula over relevant variables X and irrelevant variables Y such that $X \cap Y = \emptyset$. A SAT solver executes enumeration, either DPLL-based [60, 61] or CDCL-based [128, 146], on F, maintaining a trail I over variables $X \cup Y$. In lines 1–16 and 23–24, we consider the CDCL-based enumeration engine with chronological backtracking of our framework [139]. Now assume unit propagation has been carried out until

Input: formula $F(X, Y)$ over variables $X \cup Y$ such that $X \cap Y = \emptyset$, trail I, decision level function δ
Output: DNF M consisting of models of F projected onto X

```
Enumerate ( $F$ )
1   $I := \varepsilon$                                          // empty trail
2   $\delta := \infty$                                          // unassign all variables
3   $M := 0$                                                    // empty DNF
4   forever do
5       $C :=$ PropagateUnits ( $F, I, \delta$ )
6       if $C \neq 0$ then                                      // conflict
7           $c := \delta(C)$                                    // conflict level
8           if $c = 0$ then
9               return $M$
10          AnalyzeConflict ( $F, I, C, c$ )
11      else if all variables in $X \cup Y$ are assigned then   // $I$ is total model
12          if $V(\text{decs}(I)) \cap X = \emptyset$ then      // no relevant decision left
13              return $M \vee \pi(I, X)$                       // record $I$ projected onto $X$
14          $M := M \vee \pi(I, X)$
15          $b := \delta(\text{decs}(\pi(I, X)))$               // highest relevant decision level
16          Backtrack ( $I, b - 1$ )                            // flip last relevant decision
17      else if Entails ( $I, F$ ) then                         // $I$ is partial model
18          if $V(\text{decs}(I)) \cap X = \emptyset$ then      // no relevant decision left
19              return $M \vee \pi(I, X)$                       // record $I$ projected onto $X$
20          $M := M \vee \pi(I, X)$
21          $b := \delta(\text{decs}(\pi(I, X)))$               // highest relevant decision level
22          Backtrack ( $I, b - 1$ )                            // flip last relevant decision
23      else
24          Decide ( $I, \delta$ )
```

Figure 18.1: Early pruning for projected model enumeration. Lines 1–16 and 23–24 list CDCL-based model enumeration with chronological backtracking. If after unit propagation no conflict occurs and not all variables are assigned, an oracle might be called to check whether I entails F (line 17). If Entails returns 1, the relevant decision literal with highest decision level might be flipped. Otherwise, a decision is taken (line 24). Notice that lines 12–16 and lines 18–22 are identical.

completion, no conflict occurred and there are still unassigned variables (line 17). The trail I already might entail F, although $F|_I \neq 1$. We can check whether I entails F by an incremental call to an "oracle" [129] Entails on I and F. If Entails returns 1, then the procedure does not need to test any total extension of I, since all of them are models of F. It can proceed and flip the relevant decision literal with highest decision level (lines 21–22). If Entails returns 0, a decision needs to be taken (line 24). Notice that lines 12–16 and lines 18–22 are identical. Our method is based on chronological backtracking and follows the scheme in our framework [139], the functions PropagateUnits() and AnalyzeConflict() are taken from our previous work [138]. Entails plays the role of an "early pruning call" to a theory solver in SMT, and F plays the role of the theory [176]. Redundant work is saved by applying unit propagation until completion before calling Entails.

Quantified entailment condition. We use quantifiers with QBF semantics, and quantified formulae are always closed. A closed QBF formula evaluates to either 1 or 0. Consider $\varphi = \forall X \forall Y\,[\,F|_I\,]$, where F is a formula over variables $X \cup Y$ and the trail I ranges over $X \cup Y$. In φ, the remaining variables $(X \cup Y) - I$ are quantified. Accordingly, by $\forall X \forall Y\,[\,F|_I\,] = 1$, we express that all possible total extensions of I satisfy F, in contrast to $F|_I = 1$, expressing syntactic equality according to Section 18.2. The latter fact implies the former, but not vice versa.

Entailment under projection. If Entails implements the notion of entailment described in Section 18.2, then by calling it on I and F, we check whether $F|_J = 1$ for all total extensions J of I, i.e., whether $\forall X \forall Y\,[\,F|_I\,] = 1$. However, since we are interested in the models of F projected onto X, it suffices to check that for each possible assignment J_X to the unassigned variables in X, there exists *one* assignment J_Y to the unassigned variables in Y such that $F|_{I'} = 1$ where $I' = I \cup J_X \cup J_Y$. In essence, we need to determine the truth of the QBF formula $\forall X \exists Y\,[\,F|_I\,]$, which, in general, might be expensive, computationally. In some cases, however, a computationally cheaper (but weaker) test might be sufficient. Entails in line 17 of Enumerate can be seen as a black box pooling four entailment tests of different strengths, which we discuss in the next section.

18.4 testing entailment

Consider the original entailment condition, $\forall X \forall Y\,[\,F|_I\,] = 1$. Now we have that $\forall X \forall Y\,[\,F|_I\,] = 1 \iff \exists X \exists Y\,[\,\neg F|_I\,] = 0$. Therefore, to check whether I entails F, a SAT solver might be called to check whether $\neg F \wedge I$ is unsatisfiable. The SAT solver returns "unsat", if and only if I entails F. This observation motivates the use of dual reasoning for testing entailment in cases where cheaper tests fail. We present four flavors of the entailment test and provide examples.

1) $F|_I = 1$ (*syntactic check*). If $F|_I = 1$, also $\forall X \forall Y\,[\,F|_I\,] = 1$, and I entails F.

2) $F|_I \approx 1$ (*incomplete check in* **P**). Alternatively, if $F|_I$ differs from 1, an incomplete algorithm might be used, to check whether $\neg F \wedge I$ is unsatisfiable, by for instance executing only unit propagation or aborting after a predefined number of decision levels.

3) $F|_I \equiv 1$ (*semantic check in* **coNP**). A SAT oracle runs on $\neg F \wedge I$ until termination. Basically, it checks the unsatisfiability of $\neg F \wedge I$, i.e., whether it holds that $\exists X \exists Y\,[\,\neg F|_I\,] = 0$. If it answers "unsat", then I entails F.

4) $\forall X \exists Y\,[F|_I\,] = 1$ (*check in* Π_2^P). A QBF oracle is called to check whether the 2QBF formula $\forall X \exists Y\,[F|_I\,]$ is 1.

Modern SAT solvers mostly work on CNFs. Thus, following our dualization approach [137], we may convert $F(X,Y)$ and $\neg F(X,Y)$ into CNF formulae $P(X,Y,S)$ and $N(X,Y,T)$, where S and T denote the variables introduced by the CNF encoding. Notice that $I \wedge \neg F$ is unsatisfiable iff $I \wedge N$ is unsatisfiable.

Table 18.1 lists four examples, which differ in the strength of the required entailment test. The first column lists the formula F, the second and third column show

Table 18.1: Examples of formulae F over relevant variables X and irrelevant variables Y. For a concise representation of formulae, we represent conjunction by juxtaposition and negation by overline. In all examples, I entails F projected onto X. The entailment tests are listed from left to right in ascending order by their strength. Here, "✓" denotes the fact that I passes the test in the column, if applied to the formula in the row.

F	X	Y	I	$=1$	≈ 1	$\equiv 1$	2QBF
$(x_1 \vee y \vee x_2)$	$\{x_1, x_2\}$	$\{y\}$	x_1	✓	✓	✓	✓
$x_1 y \vee \overline{y} x_2$	$\{x_1, x_2\}$	$\{y\}$	$x_1 x_2$		✓	✓	✓
$x_1(\overline{x_2}\,\overline{y} \vee \overline{x_2} y \vee x_2 \overline{y} \vee x_2 y)$	$\{x_1, x_2\}$	$\{y\}$	x_1			✓	✓
$x_1(x_2 \leftrightarrow y)$	$\{x_1, x_2\}$	$\{y\}$	x_1				✓

the definitions of X and Y. For a concise representation of formulae, we represent conjunction by juxtaposition and negation by overline. The fourth column contains the current trail I. The fifth to eighth column denote the tests, in ascending order by their strength: $F|_I = 1$, $F|_I \approx 1$, $F|_I \equiv 1$, $\forall X \exists Y\, [F|_I] = 1$. In all examples, I entails F, and "✓" denotes the fact that I passes the test in the column, if applied to the formula in the row.

Consider the first example, $F = (x_1 \vee y \vee x_2)$ and $I = x_1$. We have $F|_I = 1$, and I entails F, which is detected by the syntactic check. For the second example, $F = x_1 y \vee \overline{y} x_2$, we have $F|_I = y \vee \overline{y}$, which is valid, but it syntactically differs from 1. The SAT solver therefore calls Entails on $\neg F \wedge I$. For $\neg F = (\overline{x_1} \vee \overline{y})(y \vee \overline{x_2})$, we find $\neg F|_I = (\overline{y})(y)$. After propagating $\overline{y}$, a conflict at decision level zero occurs, hence Entails returns 1, and an incomplete test is sufficient. In this example, $\neg F$ is already in CNF. The key idea conveyed by it can easily be lifted to the case where additional variables are introduced by the CNF transformation of $\neg F$. For the third example, $F = x_1(\overline{x_2}\,\overline{y} \vee \overline{x_2} y \vee x_2 \overline{y} \vee x_2 y)$, both $P|_I$ and $N|_I$ are undefined and contain no units. However, $N|_I$ is unsatisfiable, the SAT oracle call on $N \wedge I$ terminates with "unsat", and Entails returns 1. Hence, this example requires at least a SAT oracle. For the last example, $F = x_1(x_2 \leftrightarrow y)$, we define

$$\begin{aligned} P &= (x_1)(s_1 \vee s_2)(\overline{s_1} \vee x_2)(\overline{s_1} \vee y)(\overline{s_2} \vee \overline{x_2})(\overline{s_2} \vee \overline{y}) \quad \text{with } S = \{s_1, s_2\} \quad \text{and} \\ N &= (\overline{x_1} \vee t_1 \vee t_2)(\overline{t_1} \vee x_2)(\overline{t_1} \vee \overline{y})(\overline{t_2} \vee \overline{x_2})(\overline{t_2} \vee y) \quad \text{with } T = \{t_1, t_2\} \end{aligned}$$

We have $P|_I \neq 1$. Neither $P|_I$ nor $N|_I$ contains a unit literal, hence the incomplete test is too weak. Assume a SAT solver is called to check unsatisfiability of $N \wedge I$, and x_2 is decided first. After propagating $\overline{t_2}$, t_1 and $\overline{y}$, a total model of N is found. The SAT solver answers "sat", and Entails returns 0. A QBF solver checking $\varphi = \forall X \exists Y\, [\, x_2 y \vee \overline{x_2}\,\overline{y}\,]$ returns 1. In fact, φ is true for $I = x_2 y$ and $I = \overline{x_2}\,\overline{y}$, and Entails answers 1. Thus, at least a QBF oracle is needed.

18.5 formalization

The algorithm listed in Figure 18.1 can be expressed by means of a formal calculus. It extends our previous calculus [139] by projection and by a generalized precondition modeling an incremental call to an oracle for checking entailment (lines 17–22

EndTrue: $(F,\ I,\ M,\ \delta) \rightsquigarrow_{\mathsf{EndTrue}} M \vee m$ if $V(\mathrm{decs}(I)) \cap X = \emptyset$ and $m \overset{\text{def}}{=} \pi(I, X)$ and $\forall X \exists Y\, [\, F|_I\,] = 1$

EndFalse: $(F,\ I,\ M,\ \delta) \rightsquigarrow_{\mathsf{EndFalse}} M$ if exists $C \in F$ and $C|_I = 0$ and $\delta(C) = 0$

Unit: $(F,\ I,\ M,\ \delta) \rightsquigarrow_{\mathsf{Unit}} (F,\ I\ell,\ M,\ \delta[\ell \mapsto a])$ if $F|_I \neq 0$ and exists $C \in F$ with $\{\ell\} = C|_I$ and $a \overset{\text{def}}{=} \delta(C \setminus \{\ell\})$

BackTrue: $(F,\ I,\ M,\ \delta) \rightsquigarrow_{\mathsf{BackTrue}} (F,\ UK\ell,\ M \vee m,\ \delta[L \mapsto \infty][\ell \mapsto b])$ if $UV \overset{\text{def}}{=} I$ and $D \overset{\text{def}}{=} \overline{\pi(\mathrm{decs}(I), X)}$ and $b + 1 \overset{\text{def}}{=} \delta(D) \leqslant \delta(I)$ and $\ell \in D$ and $b = \delta(D \setminus \{\ell\}) = \delta(U)$ and $m \overset{\text{def}}{=} \pi(I, X)$ and $K \overset{\text{def}}{=} V_{\leqslant b}$ and $L \overset{\text{def}}{=} V_{>b}$ and $\forall X \exists Y\, [\, F|_I\,] = 1$

BackFalse: $(F,\ I,\ M,\ \delta) \rightsquigarrow_{\mathsf{BackFalse}} (F,\ UK\ell,\ M,\ \delta[L \mapsto \infty][\ell \mapsto j])$ if exists $C \in F$ and exists D with $UV \overset{\text{def}}{=} I$ and $C|_I = 0$ and $c \overset{\text{def}}{=} \delta(C) = \delta(D) > 0$ such that $\ell \in D$ and $\overline{\ell} \in \mathrm{decs}(I)$ and $\overline{\ell}|_V = 0$ and $F \wedge \overline{M} \models D$ and $j \overset{\text{def}}{=} \delta(D \setminus \{\ell\})$ and $b \overset{\text{def}}{=} \delta(U) = c - 1$ and $K \overset{\text{def}}{=} V_{\leqslant b}$ and $L \overset{\text{def}}{=} V_{>b}$

DecideX: $(F,\ I,\ M,\ \delta) \rightsquigarrow_{\mathsf{DecideX}} (F,\ I\ell^d,\ M,\ \delta[\ell \mapsto d])$ if $F|_I \neq 0$ and $\mathrm{units}(F|_I) = \emptyset$ and $\delta(\ell) = \infty$ and $d \overset{\text{def}}{=} \delta(I) + 1$ and $V(\ell) \in X$

DecideY: $(F,\ I,\ M,\ \delta) \rightsquigarrow_{\mathsf{DecideY}} (F,\ I\ell^d,\ M,\ \delta[\ell \mapsto d])$ if $F|_I \neq 0$ and $\mathrm{units}(F|_I) = \emptyset$ and $\delta(\ell) = \infty$ and $d \overset{\text{def}}{=} \delta(I) + 1$ and $V(\ell) \in Y$ and $X - I = \emptyset$

Figure 18.2: Rules for Enumerate.

in function Enumerate). Notably, in our work [139], only total models are found, while entailment in our actual work enables the detection of partial models. The variables in Y and S (from the CNF encoding) are treated equally with respect to unit propagation and decision. We therefore merge those two variable sets into Y to simplify the formalization. This does not affect the outcome of the entailment test. In favor of a concise description of the rules, we emphasize the differences to our previous framework [139] and refer to this work for more details.

The procedure terminates as soon as either a conflict at decision level zero occurs (rule EndFalse) or a possibly partial model is found and I contains no relevant decision literal (rule EndTrue). Requiring that no relevant decision is left on the trail prevents the recording of redundant models. The projection of I onto X is recorded. Rule Unit remains unchanged except for the missing precondition $F|_I \neq 1$. If I entails F and contains relevant decision literals, the one at the highest decision level is flipped, and the projection of I onto X is recorded (rule BackTrue).

Requiring that the last relevant decision literal is flipped prevents the recording of redundant models. Rule BackFalse remains unchanged. A decision is taken whenever $F|_I \neq 0$ and $F|_I$ contains no unit. Relevant variables are prioritized (rule DecideX) over irrelevant ones (rule DecideY).

Although not mandatory for correctness, the applicability of rule Unit might be restricted to the case where $F|_I \neq 1$. Similarly, a decision might be taken only if I does not entail F. Notice that in rules Unit, DecideX, and DecideY, the precondition $F|_I \neq 0$ can also be omitted.

18.6 CONCLUSION

In many applications (projected) partial models play a central role. For this purpose, we have presented an algorithm and its formalization inspired by how theory solvers and the SAT solver interact in SMT. The basic idea was to detect partial assignments entailing the formula on-the-fly. We presented entailment tests of different strength and computational cost and discussed examples.

The syntactic check "$F|_I = 1$" is cheapest, using clause watches or counters for keeping track of the number of satisfied clauses or alternatively the number of assigned variables (line 11 in Figure 18.1). It is also weakest, since $F|_I$ must syntactically coincide with 1. The incomplete check, denoted by "$F|_I \approx 1$", is slightly more involved. It calls a SAT solver on the negation of the formula, restricted, e.g., to unit propagation or a limited number of decision levels, and also might return "unknown". The SAT oracle executes an unsatisfiability check of $\neg F \wedge I$, given a (partial) assignment I, which might be too restrictive. The QBF oracle is the most powerful test, but also the most expensive one. It captures entailment under projection in a precise manner expressed by $\forall X \exists Y\, [\, F|_I\,] = 1$. Combining dual reasoning with oracle calls allows to avoid shrinking of total models. Finally, chronological CDCL renders the use of blocking clauses superfluous.

We claim that this is the first method combining dual reasoning and chronological CDCL for partial model detection. It is anticipated that applications with short partial models benefit most, since oracle calls might be expensive. We plan to implement our method and validate its competitiveness on applications from weighted model integration and model counting with or without projection. We also plan to investigate methods concerning the implementation of QBF oracles required by flavor 4), e.g., dependency schemes introduced by Samer and Szeider [168] or incremental QBF solving proposed by Lonsing and Egly [122].

19

DISCUSSION OF PAPER 5

The main contributions are pointed out in Section 19.1. Our goal was to detect shorter models than in our previous work [139] presented in Chapter 16. However, the note in Section 18.5 saying "Notably, in our work [139], only total models are found" requires some explanation, since we have seen in Example 16.2 and Section 17.2 that our calculus [139] is indeed able to detect partial models. We take a closer look at model detection in Section 19.2.

19.1 MAIN CONTRIBUTIONS

We present an algorithm and its formalization for enumerating the (partial) models of a propositional formula. Our framework is based on chronological CDCL and is therefore exempt from blocking clauses. It extends our previous calculus [139] by projection and by a generalized precondition based on the notion of logical entailment precised by Sebastiani [177] in the context of SAT solving. Our model enumerator takes as input a propositional formula $F(X,Y)$ defined over the disjoint sets of relevant and irrelevant variables X and Y, respectively. It runs chronological CDCL, but before taking a decision it checks whether the current partial assignment logically entails the input formula.

We are interested in the models of $F(X,Y)$ projected onto X, i. e., the precise entailment condition is given by $\forall X \exists Y [F|_I]$, which is in Π_2^P and requires an incremental QBF oracle call. This is computationally expensive, and we provide three cheaper alternatives, which sometimes might be sufficient. The first test is purely syntactic. It consists in computing the residual $F|_I$, similarly to our previous work, and similarly, to the SMT solver SPASS-SATT [33, 34], which checks whether all clauses in F are satisfied by I before taking a decision. The second and third test involve checking whether $\neg F \wedge I$ is unsatisfiable, in which case I is a model of F. The second test is incomplete and might return "unknown". It consists in, e. g., executing unit propagation on $\neg F \wedge I$, or aborting after a predefined number of decision levels and is in **P**. In the third test, the SAT oracle is run on $\neg F \wedge I$ until termination. This test is therefore in **coNP**. The fourth and most expensive test is also the most powerful one, since it captures the precise entailment condition under projection. It is the one mentioned above requiring a QBF oracle.

Logical entailment enables the detection of short partial models, which is beneficial in several ways: a larger portion of the search space is pruned, model output requires less time, and storing models requires less space. It further enables a more compact representation of the input formula than our previous methods. Our motivation is given by applications in which the size of the detected models is crucial, and in Section 19.2, we are going to elaborate on model detection.

19.2 MODEL DETECTION

In this section, the models detected by applying different entailment checks are highlighted and qualitatively compared with the models obtained in our previous work on dual projected model counting [137], referred to as "dual framework", and chronological CDCL for model counting [139], called "chronological framework". The rules are shown in Figure 11.2 and Figure 16.3. We consider a propositional formula $F(X, Y)$ defined over the set of relevant variables X and the set of irrelevant variables Y and am interested in the models of $F(X, Y)$ projected onto X. For the dual calculus, we call $P(X, Y, S)$ and $N(X, Y, T)$ the CNF transformations of F and its negation $\neg F$, respectively, where S and T denote the disjoint sets of auxiliary variables introduced by the CNF transformation.

1) $F|_I = 1$. This test corresponds to the satisfiability check in the precondition of rules EP1, BP1Fand BP1Lin our dual framework. Also, the check whether a conflict occurred is purely syntactical, since $0 \in F|_I$ can be expressed as $F|_I = 0$. Remember that in our dual framework, according to Equation 11.5 any assignment I falsifying $N(X, Y, T)$ can be extended to a model of $P(X, Y, S)$. However, this does not mean that $P(X, Y, S)|_I = 1$. Therefore, the models of our dual framework only correspond to the ones detected by the first flavor, if they are found by means of the rules EP1, BP1Fand BP1Lor if P and N are defined over the same sets of variables, as in #DPLL [24], which could be considered a special case of our dual framework [137]. In our chronological framework, assuming F is in CNF and $Y = \emptyset$, the same syntactic check is executed in both rules EndTrue and BackTrue, and the same models are found by our chronological framework and Enumerate if in place of the precise entailment check always the first flavor is executed. Our chronological framework uses exclusively syntactic satisfiability checks, and for the next three flavors, we only consider our dual framework .

2) $F|_I \approx 1$. If $\exists X \exists Y [\neg F] = 0$ or, stated otherwise, we have that $\neg F \wedge I$ is unsatisfiable, we know that I is a model of F. Obviously, this check requires an incremental call to a SAT oracle. However, there might be cases where, e. g., a conflict in $\neg F$ is obtained by adopting exclusively unit propagation. Since the conflict level is zero, $\neg F \wedge I$ is unsatisfiable, and I logically entails F. Similarly, one could run the SAT oracle on $\neg F \wedge I$ for a predefined number of decision levels, hoping that this is sufficient to determine its unsatisfiability. In our dual framework, it can happen that a conflict in N is obtained at decision level zero. Consider the formula $F(X, Y) = (x \wedge y) \vee (\neg y \wedge x)$, where $X = \{x\}$ and $Y = \{y\}$. We have $P(X, Y, S) = (\neg s_1 \vee x) \wedge (\neg s_1 \vee y) \wedge (\neg s_2 \vee \neg y) \wedge (\neg s_2 \vee x) \wedge (s_1 \vee s_2)$ and $N(X, Y, T) = (\neg x \vee \neg y) \wedge (\neg y \vee x)$ with $S = \{s_1, s_2\}$ and $T = \emptyset$. Suppose x is decided by means of the rule DecideX. The resulting trail $I = x^d$ entails F, since both xy and $x\neg y$ are models of F. Now $P|_I = (\neg s_1 \vee y) \wedge (\neg s_2 \vee \neg y) \wedge (s_1 \vee s_2)$, which is undefined, but $N|_I = (\neg y) \wedge (y)$. A unit literal with variable in $X \cup Y$ is propagated in N by deciding its negation, and a decision level is introduced. Immediately a conflict in N is obtained, but at decision level one and not zero. Only dual variables, e. g., variables in T, are propagated in N without introducing a new decision level, and only the case where a conflict occurs by exclusively propagating dual variables in N is comparable to the second flavor.

3) $F|_I \equiv 1$. It is straightforward to see that the third flavor is more powerful than our dual counting framework: Consider the example in the third line of Table 18.1. We have $F(X,Y) = (x_1) \land (\neg x_2 \lor \neg y) \land (\neg x_2 \lor y) \land (x_2 \lor \neg y) \land (x_2 \lor y)$ defined over the set or relevant variables $X = \{x_1, x_2\}$ and the set of irrelevant variables $Y = \{y\}$. Its negation is $\neg F = (\neg x_1) \lor (x_2 \land y) \lor (x_2 \land \neg y) \lor (\neg x_2 \land y) \lor (\neg x_2 \land \neg y)$, and the CNF transformations of $F(X,Y)$ and $\neg F(X,Y)$ are

$$\begin{aligned} P(X,Y,S) = {} & (x_1) \land (s_1 \lor s_2 \lor s_3 \lor s_4) \land \\ & (\neg s_1 \lor \neg x_2) \land (\neg s_1 \lor \neg y) \land (\neg s_2 \lor \neg x_2) \land (\neg s_2 \lor y) \land \\ & (\neg s_3 \lor x_2) \land (\neg s_3 \lor \neg y) \land (\neg s_4 \lor x_2) \land (\neg s_4 \lor y) \\ N(X,Y,T) = {} & (\neg x_1 \lor x_2 \lor y) \land (\neg x_1 \lor x_2 \lor \neg y) \land \\ & (\neg x_1 \lor \neg x_2 \lor y) \land (\neg x_1 \lor \neg x_2 \lor \neg y) \end{aligned}$$

with $S = \{s_1, s_2, s_3, s_4\}$ and $T = \emptyset$. Assume the current trail is $I = x_1{}^d$. Now

$$\begin{aligned} P(X,Y,S)|_I = {} & (s_1 \lor s_2 \lor s_3 \lor s_4) \land \\ & (\neg s_1 \lor \neg x_2) \land (\neg s_1 \lor \neg y) \land (\neg s_2 \lor \neg x_2) \land (\neg s_2 \lor y) \land \\ & (\neg s_3 \lor x_2) \land (\neg s_3 \lor \neg y) \land (\neg s_4 \lor x_2) \land (\neg s_4 \lor y) \\ N(X,Y,T)|_I = {} & (x_2 \lor y) \land (x_2 \lor \neg y) \land (\neg x_2 \lor y) \land (\neg x_2 \lor \neg y) \end{aligned}$$

Our dual framework now has to take a decision and continue. The remaining steps do not matter. The quintessence is that it can not find the model x_1: any model containing x_1 it detects, consists of at least two literals. Therefore, the third flavor is stronger than our dual framework.

4) $\forall X \exists Y\, [F|_I = 1]$. A similar argument shows that the dual framework in general detects models which are longer than the ones found by the fourth flavor. Consider the formula $F(X,Y,S) = x_1 \land (x_2 \leftrightarrow y)$ with $X = \{x_1, x_2\}$ and $Y = \{y\}$ shown in the last row of Table 18.1. The CNF transformations of $F(X,Y)$ and $\neg F(X,Y)$ are

$$\begin{aligned} P(X,Y,S) = {} & (x_1) \land (s_1 \lor s_2) \land (\neg s_1 \lor x_2) \land (\neg s_1 \lor y) \land \\ & (\neg s_2 \lor \neg x_2) \land (\neg s_2 \lor y) \\ N(X,Y,T) = {} & (\neg x_1 \lor t_1 \lor t_2) \land (\neg t_1 \lor x_2) \land (\neg t_1 \lor \neg y) \land \\ & (\neg t_2 \lor \neg x_2) \land (\neg t_2 \lor y) \end{aligned}$$

with $S = \{s_1, s_2\}$ and $T = \{t_1, t_2\}$. We suppose the trail to be $I = x_1{}^d$. The residuals of $F(X,Y,S)$ and $N(X,Y,T)$ are

$$\begin{aligned} P(X,Y,S)|_I &= (s_1 \lor s_2) \land (\neg s_1 \lor x_2) \land (\neg s_1 \lor y) \land (\neg s_2 \lor \neg x_2) \land (\neg s_2 \lor y) \\ N(X,Y,T)|_I &= (t_1 \lor t_2) \land (\neg t_1 \lor x_2) \land (\neg t_1 \lor \neg y) \land (\neg t_2 \lor \neg x_2) \land (\neg t_2 \lor y) \end{aligned}$$

As in the third flavor, there is no way for our dual framework to find the model x_1, since a second decision need be taken, and therefore each model containing x_1 has at least length two. The point is that in our framework we do not make oracle calls, and the models such an oracle would find is recorded explicitly in our dual framework. Therefore, it can not find the models returned by the fourth flavor.

20

PAPER 6: ON ENUMERATING SHORT PROJECTED MODELS

SUBMITTED. Preprint in: *CoRR abs/2110.12924 (2021)*. arXiv: 2110.12924. URL: https://arxiv.org/abs/2110.12924.[1]

AUTHORS. Sibylle Möhle and Roberto Sebastiani and Armin Biere.

ACKNOWLEDGMENTS. The work was supported by the LIT Secure and Correct Systems Lab funded by the State of Upper Austria, by the QuaSI project funded by D-Wave Systems Inc., and by the Italian Association for Artificial Intelligence (AI*IA). We also thank Mathias Fleury for a number of helpful discussions, which in particular led to a more elegant equivalence proof compared to our original version.

ABSTRACT. Propositional model enumeration, or All-SAT, is the task to record all models of a propositional formula. It is a key task in software and hardware verification, system engineering, and predicate abstraction, to mention a few. It also provides a means to convert a CNF formula into DNF, which is relevant in circuit design. While in some applications enumerating models multiple times causes no harm, in others avoiding repetitions is crucial. We therefore present two model enumeration algorithms, which adopt dual reasoning in order to shorten the found models. The first method enumerates pairwise contradicting models. Repetitions are avoided by the use of so-called blocking clauses, for which we provide a dual encoding. In our second approach we relax the uniqueness constraint. We present an adaptation of the standard conflict-driven clause learning procedure to support model enumeration without blocking clauses. Our procedures are expressed by means of a calculus and proofs of correctness are provided.

20.1 INTRODUCTION

The *satisfiability problem of propositional logic (SAT)* consists in determining whether for a propositional formula there exists an assignment to its variables which evaluates the formula to true, and which we call *satisfying assignment* or *model*. For proving that a formula is satisfiable, it is sufficient to provide one single model. However, sometimes determining satisfiability is not sufficient but all models are required. *Propositional model enumeration (All-SAT)*[2] is the task of enumerating (all) satisfying assignments of a propositional formula. It is a key task in, e.g.,

1 This chapter contains fixes of several minor issues on-top of the version available on arXiv.

2 For the sake of readability, we use the term *All-SAT* also if not all models are required since in principle such an algorithm could always be extended to determine all models.

bounded and unbounded model checking [21, 103, 131, 132, 184, 185], image computation [94, 95, 119, 180], system engineering [187], predicate abstraction [118], and lazy Satisfiability Modulo Theories [176].

Model enumeration also provides a means to convert a formula in Conjunctive Normal Form (CNF) into a logically equivalent formula in Disjunctive Normal Form (DNF) composed of the models of the CNF formula. This conversion is used in, e. g., circuit design [134] and has also been studied from a computational complexity point of view [195]. If the models found are pairwise contradicting, the resulting DNF is a Disjoint Sum-of-Product (DSOP) formula, which is relevant in circuit design [17, 136], and whose models can be enumerated in polynomial time [139] by simply returning their disjuncts. If the models found are not pairwise contradicting, the resulting formula is still a DNF but does not support polytime model counting. Our model enumeration algorithm basically executes a CNF to DNF conversion, and from this point of view, it can be interpreted as a knowledge compilation algorithm.

The aim of knowledge compilation is to transform a formula into another language[3] on which certain operations can be executed in polynomial time [39, 59]. This can be done, for instance, by recording the trace of an exhaustive search [100, 114, 148], and the target language in these approaches is the deterministic Decomposable Normal Form (d-DNNF),[4] which was applied, for instance, in planning [162]. In contrast, in our work we record the models of the input formula, and the resulting formula is in d-DNNF only if the detected models are pairwise contradicting.

Enumerating models requires to process the search space exhaustively and is therefore a harder task than determining satisfiability. However, since state-of-the-art SAT solvers are successfully applied in industrial applications, it seems natural to use them as a basis for model enumeration. Modern SAT solvers implement *conflict-driven clause learning (CDCL)* [127, 128, 146] with *non-chronological backtracking*.[5] If a CDCL-based SAT solver is extended to support model enumeration, adequate measures need be taken to avoid enumerating models multiple times as demonstrated by the following small example.

Example 20.1 (Multiple model enumeration)**.** *Consider the propositional formula*

$$F = \underbrace{(a \lor c)}_{C_1} \land \underbrace{(a \lor \neg c)}_{C_2} \land \underbrace{(b \lor d)}_{C_3} \land \underbrace{(b \lor \neg d)}_{C_4}$$

which is defined over the set of variables $V = \{a, b, c, d\}$. Its total models are $\mathsf{models}(F) = \{a\,b\,c\,d, a\,b\,c\,\neg d, a\,b\,\neg c\,d, a\,b\,\neg c\,\neg d\}$. These models may be represented by $a\,b$, i. e., they are given by all total extensions of $a\,b$.

Let our model enumerator be based on CDCL with non-chronological backtracking. Assume we first decide a, i. e., assign a the value true, and then b. This (partial) assignment $a\,b$ is a model of F. As in our previous work on propositional model counting [139], we flip the second decision literal, i. e., assign b the value false, in order to explore the second branch, upon which the literal d is forced to true in order to satisfy clause C_3. The resulting assignment $a\,\neg b\,d$ now falsifies clause C_4, i. e., sets all its literal to false. Conflict

3 A language in this context refers one of the various forms a formula can be expressed in, e. g., CNF and DNF denote the languages we are mostly interested in in this article.

4 A formula is in d-DNNF, if (1) the sets of variables of the conjuncts of each conjunction are pairwise disjoint, and (2) the disjuncts of each disjunction are pairwise contradicting [59].

5 Also referred to as *backjumping* in the literature.

analysis yields the unit clause $C_5 = (b)$, which is added to F. The enumerator then backtracks to decision level zero, i. e., unassigns d, b and a, and propagates b with reason C_5. No literal is enforced by the assignment b, and a decision need be taken. If we choose a, F is satisfied. The model found is $b\,a$, which is the one we had found earlier.

Multiple model enumeration in Example 20.1 is caused by the fact that after conflict analysis the same satisfying assignment is repeated, albeit in reverse order. More generally, the same satisfying assignment might be found again if the enumerator backtracks past a flipped decision literal. Avoiding enumerating models multiple times is crucial in, e. g., weighted model counting (WMC) [48, 63, 76, 174] and Bayesian inference [12], which require enumerating the models in order to compute their weight or probability. Another example is weighted model integration (WMI) [142, 143] which generalizes WMC for hybrid domains. In some applications, repeating models might lead to inefficiency and harm scalability [187]. In the context of model counting but also relevant in model enumartion, Bayardo and Pehoushek [15] identified the need for good learning similarly to its learning counterpart in CDCL, and various measures have therefore been proposed to avoid the multiple enumeration of models.

One possibility is to rule out a model which was already found by adding a *blocking clause* to the formula [131, 137, 145], which in essence is the negation of the model or the decision literals in the model to be blocked [145]. Whenever a satisfying assignment is repeated, the clause blocking it is falsified, and thus this model is not enumerated again. As soon as all models are found and the relevant blocking clauses added, the formula becomes unsatisfiable. However, there might be an exponential number of models and adding a blocking clause for each of them might result in a significant negative impact on the enumerator performance. In these cases, multiple model enumeration need be prevented by other measures. Toda and Soh [191] address this issue by adopting a variant of conflict analysis which is inspired by Gebser et al. [84] and is exempt from blocking clauses.

The use of blocking clauses can also be avoided by adopting the Davis-Putnam-Logemann-Loveland (DPLL) algorithm [60]. In DPLL, after a conflict or a model the last decision literal is flipped causing the solver to find only pairwise contradicting models. This idea was applied in the context of model counting by Birnbaum and Lozinskii [27] but can readily be adapted to support model enumeration. Chronological backtracking in Grumberg et al. [94] and in part in Gebser et al. [84] ensures that the search space is traversed in a systematic manner similarly to DPLL, and that the use of blocking clauses can be avoided. An apparent drawback of DPLL-based solvers, however, is that they might spend a significant amount of time in regions of the search space having no satisfying assignments, since—unlike CDCL-based solvers—they lack the possibility to escape those regions early.

This last issue can be addressed by the use of chronological CDCL introduced by Nadel and Ryvchin [138, 149]. Chronological CDCL combines the power of conflict-driven clause learning with chronological backtracking. Specifically, after finding a model, the last open (left) decision literal is flipped in order to process neighboring regions of the search space, while in case of a conflict, the solver is able to escape regions without solution early. In our earlier work [139], we developed a calculus for propositional model counting based on chronological CDCL and provided a proof of its correctness. We took a model enumeration approach making our method readily applicable in the context of model enumeration without repetition. However, while finding short models is crucial in, e. g., weighted model integration [142, 143], in an implementation only total models[6] might be detected as is usual in CDCL-based SAT solvers. The reason is a simple one.

To detect when a partial assignment[7] is a model of the input formula, the SAT solver would have to carry out satisfiability checks before every decision, as done by Birnbaum and Lozinskii [27]. These satisfiability checks are expensive, and assigning the remaining variables instead is more efficient, computationally. If all variables are assigned and no conflict has occurred, the SAT solver knows to have found a model. This makes sense in SAT solving. Model enumeration, however, is a harder task, and therefore more expensive methods might pay off.

One such method is *dual reasoning* [24, 137]. Our dual model counter Dualiza[8] takes as input the formula under consideration together with its negation. The basic idea is to execute CDCL on both formulae simultaneously maintaining one single trail. Whenever a conflict in the negated formula occurs, the current (partial) assignment is a model of the formula. Although developed for model counting, its adaptation for model enumeration is straightforward.

Another idea enabling the detection of short models was to check whether all total extensions of the current (partial) assignment evaluate the input formula to true before taking a decision, i. e., whether the current assignment logically entails the input formula [141, 177].

Partial assignments evaluating the input formula to true represent sets of total models of the input formula. However, these sets might not be disjoint, as is demonstrated by the following example.

Example 20.2 (Short redundant models)**.** *Let $F = (a \wedge b) \vee (a \wedge c)$ be a propositional formula defined over the set of variables $V = \{a, b, c\}$. Notice that F is not in CNF and significantly differs from the one in our previous example. Its total models are $\mathsf{models}(F) = \{a\,b\,c, a\,b\,\neg c, a\,\neg b\,c\}$. These models may also be represented by the two partial models $a\,b$ and $a\,c$. The former represents $a\,b\,c$ and $a\,b\,\neg c$, whereas the latter represents $a\,b\,c$ and $a\,\neg b\,c$. Notice that $a\,b\,c$ occurs twice.*

Partial assignments evaluating the input formula to true result in blocking clauses which are shorter than the ones blocking one single total model. Adding short blocking clauses has a twofold effect. First, a larger portion of the search space is ruled out. Second, fewer blocking clauses need be added which mitigates their negative impact on solver performance. Also, short blocking clauses generally propagate more eagerly than long ones. The need for shrinking or minimizing models has been pointed out by Bayardo and Pehoushek [15] and addressed further [10, 103, 165]. Notice that with blocking clauses CDCL can be used as in SAT solving, while in the absence of blocking clauses it need be adapted.

The reason is as follows. If a CDCL-based SAT solver encounters a conflict, it analyzes it and *learns* a clause[9] in order to prevent the solver from repeating the assignment which caused the conflict. This clause is determined by traversing the trail in reverse assignment order and resolving the reasons of the literals on the trail, starting with the conflicting clause, until the resolvent contains one single literal at the maximum decision level. If a model is found, the last decision literal is flipped in order to explore another branch of the search space. This leads to issues if this literal is encountered in later conflict analysis and no blocking clause was added, since in this case it is neither a decision literal nor a propagated literal.

6 In total models all variables occur.
7 In a partial assignment not all variables occur.
8 `https://github.com/arminbiere/dualiza`
9 We say that a clause is learned if it is added to the formula.

To address this issue, Grumberg et al. [94] introduce sub-levels for flipped decision literals treating them similarly to decision literals in future conflict analysis. Similarly to Gebser et al. [84], Toda and Soh [191] limit the level to which the solver is allowed to backtrack. These measures also ensure that enumerating overlapping partial models is avoided. However, in applications where repetitions cause no harm, the power of finding even shorter models representing larger, albeit not disjoint, sets of models, can be exploited. Shorter models are also obtained in the case of model enumeration under *projection*.

If not all variables are relevant in an application, we project the models of the input formula onto the relevant variables, or, otherwise stated, we existentially quantify the irrelevant variables. Projection occurs in, e. g., model checking [184, 185], image computation [94, 95], quantifier elimination [32, 203], and predicate abstraction [118]. The breadth of these applications highlights the relevance of projection in practice.

In this article we address the task of enumerating short projected models with and without repetition. We start by presenting a CDCL-based algorithm for the case where only pairwise contradicting, i. e., *irredundant*, models are sought. Multiple model enumeration is prevented by the addition of blocking clauses to the input formula, and dual reasoning is adopted for model shrinking. To ensure correctness of the latter, we introduce the concept of *dual blocking clauses*, which provides a solution to an issue identified in our earlier work [137]. Dual reasoning in model shrinking enables us to obtain short models, and CDCL lets us exploit the strengths of state-of-the-art SAT solvers. We express our algorithm by means of a formal calculus and provide a correctness proof.

We then present a relaxed version of our algorithm for enumerating non-contradicting, i. e., *redundant*, models. This method is exempt of blocking clauses, and consequently decision literals which were flipped after a model lack a reason. To fix this issue, we introduce an adaptation of CDCL for SAT to All-SAT. We discuss the changes to our previous algorithm needed in order to support redundant model enumeration. In the calculus, one rule is affected. We present the adapted rule and point out the relevant changes in the proofs.

This article builds on our work presented at the 23rd International Conference on Theory and Applications of Satisfiability Testing (SAT) 2020 [141]. It also uses concepts introduced by Sebastiani [177], and presented at the Second Young Scientist's International Workshop on Trends in Information Processing (YSIP2) 2017 [24] and the 30th International Conference on Tools with Artificial Intelligence (ICTAI) 2018 [137], the 22nd International Conference on Theory and Applications of Satisfiability Testing (SAT) 2019 [138], and the 5th Global Conference on Artificial Intelligence (GCAI) 2019 [139].

STRUCTURE OF THE PAPER. In Section 20.2 we give an overview over our contributions, before we introduce our notation and basic concepts needed in Section 20.3. The definitions of soundness and completeness adopted in SAT and their interpretation for model enumeration are given in Section 20.4. Dual reasoning is applied for shrinking models in Section 20.5, and an according dual encoding of blocking clauses is introduced in Section 20.6. After presenting our algorithm for projected model enumeration without repetition in Section 20.7 and providing a formalization and correctness proof and a generalization to the detection of partial models in Section 20.8, we turn our attention to projected model enumeration with repetition. We adapt CDCL for SAT to support conflict analysis in the context of model enumeration without the use of blocking clauses in Section 20.9 and discuss the changes to our method needed to support multiple model enumeration in Section 20.10, before we conclude in Section 20.11.

20.2 OVERVIEW OF CONTRIBUTIONS

In this section, we give a high-level overview over our contributions.

20.2.1 *Correctness with Respect to Model Enumeration*

We recall the definitions of soundness and completeness in the context of SAT solving according to Weidenbach [197], before stating them for All-SAT.

20.2.2 *Model Shrinking*

Obtaining short models is our main aim in this work. To this end, in our model enumeration algorithms we adopt a dual method for model shrinking.

Basically, whenever a model is found, a second SAT solver is called incrementally on this model and the negation of the formula. A conflict is obtained and conflict analysis executed to determine the literals involved in the conflict, which constitutes the shortened model.

20.2.3 *Irredundant Model Enumeration Under Projection*

We present a CDCL-based algorithm for projected model enumeration without repetition using dual model shrinking and blocking clauses. In a dual setting, models need be blocked not only in the formula but in its negation as well. A model is blocked in the negated formula by disjoining the two. However, the resulting formula is no longer in CNF. To address this issue, we present a dual encoding for blocking clauses.

The shrunken model is then used to determine the backtrack level, which might be much smaller than the current one. Conflict analysis and unit propagation are executed as in CDCL. The blocking clauses ensure no problems arise in conflict analysis. Our algorithm is expressed by means of a calculus whose rules cover termination, backtracking, unit propagation and decisions.

We identify three invariants and show that they hold in all non-terminal states, that our system always makes progress, and that it eventually terminates. Equivalence of the resulting formula and the input formula projected onto the relevant variables proves soundness and completeness and concludes our proof.

A generalization of our algorithm to the case where partial satisfying assignments are found is discussed. This generalization makes sense since we do not guarantee that our model shrinking method gives us the minimal model.

20.2.4 *Redundant Model Enumeration Under Projection*

This method uses dual model shrinking. It is exempt of blocking clauses, and the conflict analysis procedure need be adapted. We propose to annotate the flipped decision literals with the clause which would be added as blocking clause and discuss the relevant changes to our algorithm, calculus, proof, and its generalization to the case where partial models are found.

20.3 PRELIMINARIES

In this section we provide the concepts and notation on which our presentation relies: propositional satisfiability (SAT) and incremental SAT solving, projection, and the dual representation of a formula, which constitutes the basis for dual reasoning.

20.3.1 *Propositional Satisfiability (SAT)*

The set containing the Boolean constants 0 (false) and 1 (true) is denoted with $\mathbb{B} = \{0, 1\}$. Let V be a set of propositional (or Boolean) variables. A *literal* is either a variable $v \in V$ or its negation $\neg v$. We write $\overline{\ell}$ to denote the *complement* of ℓ assuming $\overline{\ell} = \neg\ell$ and $\neg\neg\ell = \ell$. The variable of a literal ℓ is obtained by $\mathsf{var}(\ell)$. This notion is extended to formulae, clauses, cubes, and sets of literals.

Most SAT solvers work on formulae in *Conjunctive Normal Form (CNF)*, which are conjunctions of *clauses*, which are disjunctions of literals. These SAT solvers implement efficient algorithms tailored for CNFs, such as unit propagation, which will be presented below. In contrast, a formula in *Disjunctive Normal Form (DNF)* is a disjunction of *cubes*, which are conjunctions of literals. We interpret formulae as sets of clauses and write $C \in F$ to refer to a clause C occurring in the formula F. Accordingly, we interpret clauses and cubes as sets of literals. The empty CNF formula and the empty cube are denoted by 1, while the empty DNF formula and the empty clause are represented by 0.

A total assignment $\sigma\colon V \mapsto \mathbb{B}$ maps V to the truth values 0 and 1. It can be applied to a formula F over a set of variables V to obtain the truth value $\sigma(F) \in \mathbb{B}$, also written $F|_\sigma$. The *value of F under* σ is denoted by $\sigma(F)$. A sequence $I = \ell_1 \ldots \ell_n$ with mutually exclusive variables ($\mathsf{var}(\ell_i) \neq \mathsf{var}(\ell_j)$ for $i \neq j$) is called a *trail*. If their variable sets are disjoint, trails and literals may be concatenated, denoted by $I = I'I''$ and $I = I'\ell I''$. We treat trails as conjunctions or sets of literals and write $\ell \in I$ if ℓ is contained in I. Trails can also be interpreted as partial assignments with $I(\ell) = 1$ iff $\ell \in I$. Similarly, $I(\ell) = 0$ iff $\neg\ell \in I$, and $I(\ell)$ is undefined iff $\mathsf{var}(\ell) \notin \mathsf{var}(I)$. The unassigned variables in V are denoted by $V - I$ and the empty trail by ε.

The literal ℓ can be either *decided* or *propagated*. In the former case, its value is chosen according to some heuristic by a *decision*, and ℓ is called *decision literal*. In the latter case, there exists a clause $C \in F$ containing ℓ in which all literals except ℓ evaluate to false under the current (partial) assignment. The literal ℓ is called *unit literal* or *unit* and C a *unit clause*. In order to satisfy C, and thus F, the literal ℓ need be assigned the value true. After being propagated, the literal ℓ becomes a *propagation literal*, and C its *reason*. The corresponding rule is the *unit propagation* rule. We annotate decision literals on the trail by a superscript, e. g., ℓ^d, denoting open "left" branches in the sense of DPLL. If a decision literal ℓ is flipped, its complement $\overline{\ell}$ opens a "right" branch. Both propagation literals and flipped decision literals are annotated on the trail by their reason, as in ℓ^C.

The trail is partitioned into blocks, called *decision levels*, which extend from a decision literal to the last literal preceding the next decision literal. Literals occurring before the first decision are assigned at decision level zero. They are assigned exclusively by unit propagation. The decision level of a variable $v \in V$ is obtained by applying the *decision level function* $\delta \colon V \mapsto \mathbb{N} \cup \infty$. We extend δ accordingly to literals ℓ, non-empty clauses C, and non-empty sequences of literals I, by defining $\delta(\ell) = \delta(\mathsf{var}(\ell))$, $\delta(C) = \max\{\delta(\ell) \mid \ell \in C\}$, and $\delta(I) = \max\{\delta(\ell) \mid \ell \in I\}$. Accordingly, we define $\delta(L) = \max\{\delta(\ell) \mid \ell \in L\}$ for a set of literals $L \neq \emptyset$. If v is unassigned, we have $\delta(v) = \infty$, and $\delta(0) = \delta(\varepsilon) = \delta(\emptyset) = 0$ for the empty clause, the empty sequence and the empty set of literals. Whenever a variable is assigned or unassigned, the decision level function δ is updated. If $\mathsf{var}(\ell)$ is assigned at decision level d, we write $\delta[\,\ell \mapsto d\,]$. If all variables in the set of variables V are unassigned, we write $\delta[\,V \mapsto \infty\,]$ or $\delta \equiv \infty$ as a shortcut. Similarly, if all literals occurring in a sequence of literals I are unassigned, we write $\delta[\,I \mapsto \infty\,] = \delta[\,\mathsf{var}(I) \mapsto \infty\,]$. The function δ is left-associative, i. e., $\delta[\,I \mapsto \infty\,][\,\ell \mapsto d\,]$ first unassigns all variables on I and then assigns literal ℓ at decision level d.

We call *residual* of F under I, denoted $F|_I$, the formula $I(F)$ obtained by assigning the variables in F their truth value. If F is in CNF, this amounts to removing from F all clauses containing a literal $\ell \in I$ and removing from the remaining clauses all occurrences of $\neg\ell$. If $F|_I = 1$, we say that I *satisfies* F or that I is a *model* of F. If all variables are assigned, we call I a *total model* of F. Following the distinction highlighted by Sebastiani [177], if I is a partial assignment, we say that I *evaluates F to* 1, written $I \vdash F$, if $F|_I = 1$, and that I *logically entails* F, written $I \models F$, or that I is a *partial model* of F, if all total assignments extending I satisfy F. Notice that $I \vdash F$ implies that $I \models F$ but not vice versa: e. g., if $F \overset{\text{def}}{=} (a \wedge b) \vee (a \wedge \neg b)$ and $I \overset{\text{def}}{=} a$, then $I \models F$ but $I \nvdash F$, because $F|_I = (b \vee \neg b) \neq 1$. If F is in CNF without valid clauses, i. e., without clauses containing contradicting literals, then $I \vdash F$ iff $F|_I = 1$. We say that I *evaluates F to* 0 or that I is a *countermodel* of F, iff $F|_I = 0$. If F is in CNF, its residual under I contains the empty clause, written $0 \in F|_I$. Similarly, we say that a *conflict* occurs and call the clause whose literals are set to false under I *conflicting clause* and its decision level *conflict level*. In CDCL with non-chronological backtracking this is the current decision level $\delta(I)$.

20.3.2 *Conflict-Driven Clause Learning*

Suppose the current trail I falsifies the formula F. The basic idea is to determine a clause, let's say D, containing the negated assignments responsible for the conflict. By adding D to F, this assignment is blocked. Moreover, backtracking to the second highest decision level in D results in D becoming unit, and its literal with highest decision level is propagated.

A main ingredient of the clause learning algorithm is *resolution* [166]. Given two clauses $(A \vee \ell)$ and $(B \vee \neg\ell)$, where A and B are disjunctions of literals and ℓ is a literal, their *resolvent* $(A \vee \ell) \otimes_\ell (B \vee \neg\ell) = (A \vee B)$ is obtained by resolving them on ℓ.

The clause D is determined by a sequence of resolution steps, which can be read off either the trail or the *implication graph*, which is defined as follows. Decision literals are represented as nodes on the left and annotated with their decision level. Propagated literals are internal nodes with one incoming arc originating from each node representing a literal in their reason. A conflict is represented by the special node κ whose incoming arcs are annotated with the conflicting clause.

Example 20.3 (Trail and implication graph). *Consider the formula*

$$F = \underbrace{(\neg a \vee b)}_{C_1} \wedge \underbrace{(\neg c \vee d)}_{C_2} \wedge \underbrace{(\neg b \vee \neg c \vee \neg d)}_{C_3}$$

over the set of variables $V = \{a, b, c, d\}$. Assume we first decide a, then propagate b with reason C_1 followed by deciding c and propagating d with reason C_2. Under this assignment, C_3 is falsified. The current trail is given by

$$I = a^d\, b^{C_1}\, c^d\, d^{C_2},$$

and the according implication graph is

$$\begin{tikzcd}
a@1 \arrow[r, "C_1"] & b \arrow[dr, bend left, "C_3"] & \\
c@2 \arrow[r, "C_2"] \arrow[rr, bend right, "C_3"'] & d \arrow[r, "C_3"] & \kappa
\end{tikzcd}$$

During conflict analysis, the conflicting clause is resolved with the reason of one of its literals. This procedure is repeated with the reason of one literal in the resolvent, and so on, until the resolvent contains one single literal at conflict level. In the following example, clause learning based on I and the implication graph is shown.

Example 20.4 (Conflict-driven clause learning). *Consider the situation in Example 20.3. The conflicting clause is $C_3 = (\neg b \vee \neg c \vee \neg d)$.*

In order do determine the sequence of resolution steps in order to learn a clause from I, we resolve the conflicting clause C_3 with the reason of the last propagated literal on I, C_2, obtaining $\neg b \vee \neg c$, which already contains only one literal at conflict level 2, namely $\neg c$.

Considering the implication graph, we can resolve C_3 with either C_2 or C_1. If we choose C_1, the resolvent $(\neg a \vee \neg c \vee \neg d)$ contains two literals at conflict level, hence resolution with C_2 is needed resulting in $(\neg a \vee \neg c)$. Notice that resolving C_3 with C_2 first would save one resolution step.

20.3.3 *Incremental SAT Solving*

The basic idea of incremental SAT solving is to exploit the progress made during the search process, if similar formulae need be solved. So, instead of discarding the learned clauses they are retained between the individual SAT calls.

Hooker [98] presented the idea of incremental in the context of knowledge-based reasoning. Eén and Sörensson [67] introduced the concept of *assumptions* in the context of incremental SAT solving, which fits our needs best. Assumptions can be viewed as unit clauses added to the formula. They basically represent a (partial) assignment whose literals remain set to true during the solving process. In particular, backtracking does not occur past any assumed literal.

20.3.4 *Projection*

We are interested in enumerating the models of a propositional formula *projected* onto a subset of its variables. To this end we partition the set of variables $V =$

$X \cup Y$ into the set of *relevant variables* X and the set of *irrelevant variables* Y and write $F(X \cup Y)$ to express that F depends on the variable set $X \cup Y$. Accordingly, we decompose the assignment $\sigma = \sigma_X \cup \sigma_Y$ into its relevant part $\sigma_X \colon X \mapsto \mathbb{B}$ and its irrelevant part $\sigma_Y \colon Y \mapsto \mathbb{B}$ following the convention introduced in our earlier work on dual projected model counting [137]. The main idea of projection onto the relevant variables is to existentially quantify the irrelevant variables. The total models of $F(X \cup Y)$ projected onto X are therefore

$$\mathsf{models}(\exists\, Y.\, F(X,Y)) = \{\tau \colon X \to \mathbb{B} \mid \text{exists } \sigma : V \to \mathbb{B} \text{ with}$$
$$\sigma\big(F(X,Y)\big) = 1 \;\text{ and }\; \tau = \sigma_X\},$$

and enumerating all models of F without projection is therefore the special case where $Y = \emptyset$. The projection of the trail I onto the set of variables X is denoted by $\pi(I, X)$ and $\pi(F(X,Y), X) \equiv \exists\, Y\, [\, F(X,Y)\,]$.

Example 20.5 (Projected models)**.** *Consider again the formula F in Example 20.1. Its unprojected total models are given by* $\mathsf{models}(F) = \{a\,b\,c\,d, a\,b\,c\,\neg d, a\,b\,\neg c\,d, a\,b\,\neg c\,\neg d\}$*. Its total models projected onto* $X = \{a, c\}$ *are* $a\,c$ *and* $a\,\neg c$*.*

In order to benefit from the efficient methods SAT solvers execute on CNF formulae, we transform an arbitrary formula $F(X,Y)$ into CNF by, e. g., the Tseitin transformation [192].[10] By this transformation, auxiliary variables, also referred to as *Tseitin* or *internal* variables, are introduced. The Tseitin transformation is *satisfiability-preserving*, i. e., a satisfiable formula is not turned into an unsatisfiable one and, similarly, an unsatisfiable formula is not turned into a satisfiable one. The Tseitin variables, which we denote by S, are defined in terms of the variables in $X \cup Y$, which we call *input variables*. As a consequence, for each total assignment to the variables in $X \cup Y$ there exists one single assignment to the variables in S such that the resulting assignment is a model of F, and therefore the model count is preserved. Due to the introduction of the Tseitin variables the resulting formula $P(X,Y,S) = \mathsf{tseitin}(F(X,Y))$ is not logically equivalent to $F(X,Y)$, i. e., $\mathsf{models}(F) \neq \mathsf{models}(P)$, and the Tseitin transformation is not *equivalence-preserving*. However, the models of $P(X,Y,S)$ projected onto the input variables are exactly the models of $F(X,Y)$, and

$$\exists\, S\, [\, P(X,Y,S)\,] \equiv F(X,Y). \tag{20.1}$$

The total models of F projected onto X are accordingly given by

$$\mathsf{models}(\exists\, Y, S\, [\, P(X,Y,S)\,]) = \mathsf{models}(\exists\, Y\, [\, F(X,Y)\,]). \tag{20.2}$$

20.3.5 *Dual Representation of a Formula*

We make use of the dual representation of a formula introduced in our earlier work [137]. Let $F(X,Y)$ and $P(X,Y,S)$ be defined as in Section 20.3.4, and let $N(X,Y,T) = \mathsf{tseitin}(\neg F(X,Y))$ be a CNF representation of $\neg F$, where T denotes the set of Tseitin variables introduced by the transformation, i. e.,

$$\exists\, T\, [\, N(X,Y,T)\,] \equiv \neg F(X,Y). \tag{20.3}$$

10 It turns out that in the context of dual projected model enumeration also the Plaisted-Greenbaum transformation [164] might be used although in general it does not preserve the model count.

The formulae $P(X,Y,S)$ and $N(X,Y,T)$ are a *dual representation*[11] of $F(X,Y)$. For the sake of readability, we also may write F, P, and N. Notice that this representation is not unique in general. Besides that, $P(X,Y,S)$ and $N(X,Y,T)$ share the set of input variables $X \cup Y$, and $S \cap T = \emptyset$. We have

$$\exists S\,[\,P(X,Y,S)\,] \equiv \neg\exists T\,[\,N(X,Y,T)\,]\,, \tag{20.4}$$

and in an earlier work [137] we showed that during the enumeration process a generalization of the following always holds assuming we first split on variables in X and then on variables in $Y \cup S$ but never on variables in T:

$$(\,\neg\exists\, T\,[\,N(X,Y,T)|_I\,]\,) \models (\,\exists\, S\,[\,P(X,Y,S)|_I\,]\,)\,, \tag{20.5}$$

where I is a trail over variables in $X \cup Y \cup S \cup T$. Obviously, also

$$(\,\exists\, S\,[\,P(X,Y,S)|_I\,]\,) \models (\,\neg\exists\, T\,[\,N(X,Y,T)|_I\,]\,)\,, \tag{20.6}$$

saying that whenever I can be extended to a model of P, all extensions of it falsify N. This property is a basic ingredient of our dual model shrinking method.

20.4 SOUNDNESS AND COMPLETENESS

In this section, we recall the definitions of soundness and completeness with respect to SAT solving following Weidenbach [197] and present their definition in the context of model enumeration. In his work, Weidenbach refers to total models, but his definitions are readily applicable for partial models as well. Accordingly, and as we are targeting the detection of partial models, our definitions apply to both total and partial models.

Definition 20.1 (Soundness in satisfiability)**.** *An algorithm for satisfiability is* sound*, iff it is guaranteed that the model it finds is a model of the formula.*

Definition 20.2 (Completeness in satisfiability)**.** *An algorithm for satisfiability is* complete*, iff it is guaranteed to find a model if the formula is satisfiable.*

Definition 20.3 (Strong completeness in satisfiability)**.** *An algorithm for satisfiability is* strongly complete *iff it is guaranteed that it finds any model of a satisfiable formula.*

Definition 20.1 says that the model returned by a sound satisfiability algorithm is a model of the formula. In the context of model enumeration, the counterpart of a model is a sequence containing all models. Hence, soundness in model enumeration states that the sequence of models returned by the model enumerator contains only models of the formula.

Definition 20.4 (Soundness in model enumeration)**.** *A model enumeration algorithm is* sound*, iff it is guaranteed that the models in the sequence of all models it enumerates are models of the formula.*

Definition 20.2 ensures that if the input formula is satisfiable, a complete satisfiability algorithm finds a model for it. In the context of model enumeration, we would expect that a complete enumeration algorithm finds a sequence of models, if the formula is satisfiable.

11 Referred to as *combined formula pair* of $F(X,Y)$ in our previous work [137].

Definition 20.5 (Completeness in model enumeration)**.** *A model enumeration algorithm is* complete*, iff it is guaranteed to find a sequence containing all models of a satisfiable formula.*

According to Definition 20.3, a strongly complete satisfiability algorithm finds *any*, i. e., arbitrary, model. For model enumeration this would mean that a strongly complete model enumeration algorithm finds *any*, i. e., arbitrary, sequence of models.

Definition 20.6 (Strong completeness in model enumeration)**.** *A model enumeration algorithm is* strongly complete*, iff it is guaranteed to find any arbitrary sequence containing all models of a satisfiable formula.*

Note: Strong completeness can be obtained by restarting after the addition of a blocking clause or according to some heuristic if no blocking clauses are used.

20.5 DUAL REASONING FOR MODEL SHRINKING

In a previous work, we adopted dual reasoning for obtaining partial models [137]. Basically, we executed CDCL on the formula under consideration and its negation simultaneously exploiting the fact that CDCL is biased towards detecting conflicts. Our experiments showed that dual reasoning detects short models. However, processing two formulae simultaneously turned out to be computationally expensive.

In another work [141] we propose, before taking a decision, to check whether the current (partial) assignment logically entails the formula under consideration. We present four flavors of the entailment check, some of which use a SAT oracle and rely on dual reasoning.

The method introduced in this work, instead, exploits the effectiveness of dual reasoning in detecting short partial models while avoiding both processing two formulae simultaneously and oracle calls, which might be computationally expensive. In essence, we let the enumerator find total models and shrink them by means of dual reasoning.

Assume our task is to determine the models of a formula $F(X, Y)$ over the set of relevant variables X and irrelevant variables Y projected onto X, and let $P(X, Y, S)$ and $N(X, Y, T)$ be CNF representations of F and $\neg F$, respectively, as introduced in Section 20.3. Obviously, Equation 20.2–Equation 20.9 hold. Suppose standard CDCL is executed on P. We denote with I the trail which ranges over variables in $X \cup Y \cup S \cup T$, where S and T are the Tseitin variables occurring in P and N, respectively.

Now assume a total model I of P is found. A second SAT solver is incrementally invoked on $\pi(I, X \cup Y) \wedge N$. Since $\pi(I, X \cup Y) \models F$ and all variables in $X \cup Y$ are assigned, due to Equation 20.6 a conflict in N occurs by propagating variables in T only. If conflict analysis is carried out as described in Section 20.3.2, the learned clause $\neg I^\star$ contains only negated assumption literals.[12] On the one hand, $\neg I^\star$ represents a cause for the conflict in N. On the other hand, due to Equation 20.5, its negation $I^\star$ represents a (partial) model of F. More precisely, $I^\star$ represents all total models of F projected onto $X \cup Y$ in which the variables in $X \cup Y$ not occurring in $I^\star$ may assume any truth value.

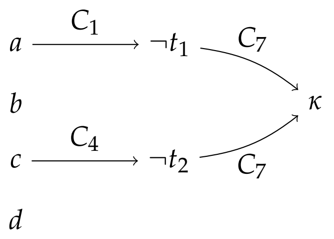
12 See also the work by Niemetz et al. [153].

Example 20.6 (Model shrinking by dual reasoning)**.** *Let $F = (a \lor b) \land (c \lor d)$ be a propositional formula over the set of variables $V = \{a, b, c, d\}$. Without loss of generalization, suppose we want to enumerate the models of F projected onto V. Assume a total model $I = a\,b\,c\,d$ has been found. We call a second SAT solver on $N \land I$, where*

$$\begin{aligned} N = {} & \underbrace{(\neg t_1 \lor \neg a)}_{C_1} \land \underbrace{(\neg t_1 \lor \neg b)}_{C_2} \land \underbrace{(a \lor b \lor t_1)}_{C_3} \land \\ & \underbrace{(\neg t_2 \lor \neg c)}_{C_4} \land \underbrace{(\neg t_2 \lor \neg d)}_{C_6} \land \underbrace{(c \lor d \lor t_2)}_{C_6} \land \\ & \underbrace{(t_1 \lor t_2)}_{C_7} \end{aligned}$$

is the Tseitin encoding of $\neg F = (\neg a \land \neg b) \lor (\neg c \land \neg d)$ with Tseitin variables $T = \{t_1, t_2\}$. The clauses C_1 to C_3 encode $(t_1 \leftrightarrow (\neg a \land \neg b))$, the clauses C_4 to C_6 encode $(t_2 \leftrightarrow (\neg c \land \neg d))$, and C_7 encodes $(t_1 \lor t_2)$.

The literals on I are considered assumed variables, annotated by, e.g., a^a, and $I = a^a\,b^a\,c^a\,d^a$. After propagating $\neg t_1$ with reason C_1 and $\neg t_2$ with reason C_4, the clause C_7 becomes empty. The current trail is $I' = a^a\,b^a\,c^a\,d^a\,\neg t_1{}^{C_1}\,\neg t_2{}^{C_4}$. We resolve C_7 with C_4 to obtain the clause $(t_1 \lor \neg c)$, which we then resolve with C_1. The resolvent is $(\neg c \lor \neg a)$, which contains only assumed literals which have no reason in I', and thus can not be resolved further. Below on the left hand side, the implication graph is visualized, and the corresponding resolution steps are depicted on the right hand side.


$$\begin{tikzcd} a \arrow[r, "C_1"] & \neg t_1 \arrow[dr, bend left, "C_7"] & \\ b & & \kappa \\ c \arrow[r, "C_4"] & \neg t_2 \arrow[ur, bend right, "C_7"'] & \\ d & & \end{tikzcd}$$

$$\dfrac{\dfrac{(t_1 \lor t_2) \qquad (\neg t_2 \lor \neg c)}{(t_1 \lor \neg c)} \qquad (\neg t_1 \lor \neg a)}{(\neg c \lor \neg a)}$$

The negation of the clause $(\neg c \lor \neg a)$, $c\,a$, is a countermodel of $\neg F$ and hence a model of F. In this case, it is also minimal w.r.t. the number of literals.

Note: The gain obtained by model shrinking is twofold. On the one hand, it enables the (implicit) exploration of multiple models in one pass: e.g., in Example 20.6, the model $c\,a$ represents four total models, namely, $a\,b\,c\,d$, $a\,b\,c\,\neg d$, $a\,\neg b\,c\,d$, and $a\,\neg b\,c\,\neg d$. On the other hand, short models result in short blocking clauses ruling out a larger part of the search space, as mentioned earlier.

20.6 DUAL ENCODING OF BLOCKING CLAUSES

Recall that we make use of Equation 20.4. If a blocking clause is added to P and N is not updated accordingly, P and N do not represent the negation of each other anymore, and Equation 20.4 ceases to hold. This might lead to multiple model enumerations in the further search. This issue can be remediated by adding the shrunken models disjunctively to N. To retain N in CNF and ensure Equation 20.4, we propose the following *dual encoding* of the blocking clauses.

We denote with $\mathsf{tseitin}()$ the function which takes as argument an arbitrary propositional formula and returns its Tseitin transformation. For the sake of readability, we write F, P, and N as well as their indexed variants instead of $F(X \cup Y)$, $P(X \cup Y \cup S)$ and $N(X \cup Y \cup T)$. We define

$$P_0 = \mathsf{tseitin}(F) \tag{20.7}$$

$$N_0 = t_0 \wedge \mathsf{tseitin}(t_0 \leftrightarrow \neg F). \tag{20.8}$$

Let I_1 be a trail such that I_1 evaluates F to true, i. e., $I_1 \vdash F$. A second SAT solver SAT is called on $\pi(I_1, X \cup Y) \wedge N_0$, and a conflict is obtained as argued above. Assume SAT returns the assignment $I_1^\star \leqslant I_1$ such that $SAT(\pi(I_1^\star, X \cup Y), N_0) =$ UNSAT. Then $\neg\pi(I_1^\star, X)$ is added to P_0 obtaining $P_1 = P_0 \wedge \neg\pi(I_1^\star, X)$. To ensure Equation 20.4, we define $N_1 = (t_0 \vee t_1) \wedge \mathsf{tseitin}(t_0 \leftrightarrow \neg F) \wedge \mathsf{tseitin}(t_1 \leftrightarrow \pi(I_1^\star, X))$, and so on.

At the n^{th} step, we have

$$P_n = P_0 \wedge \bigwedge_{i=1}^{n} \neg\pi(I_i^\star, X) \tag{20.9}$$

$$N_n = (t_0 \vee \bigvee_{i=1}^{n} t_i) \wedge \mathsf{tseitin}(t_0 \leftrightarrow \neg F) \wedge \bigwedge_{i=1}^{n} \mathsf{tseitin}(t_i \leftrightarrow \pi(I_i^\star, X)), \tag{20.10}$$

where the red parts denote the additions to P_0 and N_0.

Let I_{n+1} be a trail evaluating P_n to true, i. e., $I_{n+1} \vdash P_n$. We invoke SAT on $\pi(I_{n+1}, X \cup Y) \wedge N_n$ leading to a conflict as described above. Assume SAT returns $I_{n+1}^\star \leqslant I_{n+1}$, such that $SAT(\pi(I_{n+1}^\star, X \cup Y), N_n) =$ UNSAT. We add $\neg\pi(I_{n+1}^\star, X)$ to P_n and update N_n accordingly. Now we have

$$P_{n+1} = P_n \wedge \neg\pi(I_{i+1}^\star, X) \tag{20.11}$$

$$N_{n+1} = N_n \setminus \{(t_0 \vee \bigvee_{i=1}^{n} t_i)\} \wedge (t_0 \vee \bigvee_{i=1}^{n+1} t_i) \wedge \mathsf{tseitin}(t_{n+1} \leftrightarrow \pi(I_{n+1}^\star, X)), \tag{20.12}$$

$$\begin{aligned} \text{where} \quad & I_{i+1} \vdash P_i \text{ for } 0 \leqslant i \leqslant n \\ \text{and} \quad & I_{i+1}^\star \leqslant I_{i+1} \text{ is s. t. } SAT(\pi(I_{i+1}^\star, X \cup Y), N_i) = \text{UNSAT}. \end{aligned}$$

Proposition 20.1. *Let $F(X, Y)$ be an arbitrary propositional formula over the relevant variables X and the irrelevant variables Y. Let F and $\neg F$ be encoded into CNFs P_0 and N_0, respectively, according to Equation 20.7 and Equation 20.8. If for all models found blocking clauses are added to P_0 and N_0 according to Equation 20.9 and Equation 20.10, then only pairwise contradicting models are found, i. e., $\pi(I_i^\star, X)$ and $\pi(I_j^\star, X)$ are pairwise contradicting for every $i \neq j$.*

Proof. By construction, $N_n \equiv \neg F \vee \bigvee_{i=1}^{n} \pi(I^\star, X \cup Y)$, and $\pi(I^\star, X \cup Y) \wedge N_n \equiv 0$. Furthermore, $\pi(I^\star, X \cup Y) \wedge \neg F \equiv 0$. We have

$$\begin{aligned}
&0 \equiv \\
&\pi(I^\star_{n+1}, X \cup Y) \wedge (\neg F \vee \bigvee_{i=1}^{n} \pi(I^\star, X \cup Y)) = \\
&(\pi(I^\star_{n+1}, X \cup Y) \wedge \neg F) \vee (\pi(I^\star_{n+1}, X \cup Y) \wedge \bigvee_{i=1}^{n} \pi(I^\star_i, X \cup Y)) \equiv \\
&(\pi(I^\star_{n+1}, X \cup Y) \wedge \bigvee_{i=1}^{n} \pi(I^\star_i, X \cup Y)) \equiv \\
&(\pi(I^\star_{n+1}, X) \wedge \bigvee_{i=1}^{n} \pi(I^\star_i, X)),
\end{aligned}$$

since $I^\star_i$ contains only relevant variables. Hence, $\pi(I^\star_{n+1}, X) \wedge \pi(I^\star_i, X) \equiv 0$ for $i = 1, \ldots, n$. □

Note: Equation 20.4 always holds:

$$\exists S\,[\,P_i(X, Y, S)\,] \equiv \neg \exists T\,[\,N_i(X, Y, T)\,] \quad \text{for all } 0 \leqslant i \leqslant n + 1.$$

Consequently, also Equation 20.5 and Equation 20.6 hold:

$$(\,\neg \exists\, T\,[\,N_i(X, Y, T)|_I\,]\,) \models (\,\exists\, S\,[\,P_i(X, Y, S)|_I\,]\,) \quad \text{for all } 0 \leqslant i \leqslant n + 1$$

$$(\,\exists\, S\,[\,P_i(X, Y, S)|_I\,]\,) \models (\,\neg \exists\, T\,[\,N_i(X, Y, T)|_I\,]\,) \quad \text{for all } 0 \leqslant i \leqslant n + 1$$

However, for our usage we may use the implication in the forward direction only and write $t_i \rightarrow \pi(I^\star_i, X)$ and $t_{n+1} \rightarrow \pi(I^\star_{n+1}, X)$ in Equation 20.10 and Equation 20.12 without compromising correctness for the following reason: The formula N_i is called always under I_{i+1} which falsifies all $I^\star_k$ for $0 \leqslant k \leqslant i$. Hence, $\pi(I^\star_i, X) \rightarrow t_i$ is always true.

Example 20.7 (Dual blocking clauses). *We clarify the proposed encoding by a small example and show that it prevents multiple model counts. Let our example be $F = x_1 \vee (x_2 \wedge x_3)$ and assume we have found the model $I^\star_1 = x_1$. Then*

$$P_1 = (\neg s_1 \vee x_2) \wedge (\neg s_1 \vee x_3) \wedge (s_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_1 \vee s_1) \wedge (\neg x_1)$$

and

$$\begin{aligned}
N_1 = &\underbrace{(\neg t_1 \vee \neg x_1)}_{C_1} \wedge \underbrace{(\neg t_1 \vee \neg x_2)}_{C_2} \wedge \underbrace{(t_0 \vee x_1 \vee x_2)}_{C_3} \wedge \\
&\underbrace{(\neg t_2 \vee \neg x_1)}_{C_4} \wedge \underbrace{(\neg t_2 \vee \neg x_3)}_{C_5} \wedge \underbrace{(t_2 \vee x_2 \vee x_3)}_{C_6} \wedge \\
&\underbrace{(t_1 \vee t_2 \vee t_3)}_{C_7} \wedge \underbrace{(\neg t_3 \vee x_1)}_{C_8} \wedge \underbrace{(t_3 \vee \neg x_1)}_{C_9}
\end{aligned}$$

where the blue parts denote the corresponding additions to P_0 and N_0. If now we find a total model $I_2 = \neg x_1\, x_2\, x_3$, we obtain a conflict in N_1 by unit propagating variables t_1, t_3, and t_3 only. The conflicting clause is $(t_1 \vee t_2 \vee t_3)$. The implication graph is depicted below on the left hand side, the corresponding resolution steps for conflict analysis below on the right hand side.

$$\begin{tikzcd}
\neg x_1 \arrow[r, "C_8"] & \neg t_3 \arrow[dr, "C_7"] & \\
x_2 \arrow[r, "C_2"] & \neg t_1 \arrow[r, "C_7"] & \kappa \\
x_3 \arrow[r, "C_5"] & \neg t_2 \arrow[ur, "C_7"'] &
\end{tikzcd}$$

$$\dfrac{\dfrac{\dfrac{(t_1 \vee t_2 \vee t_3) \qquad (\neg t_2 \vee \neg x_3)}{(t_1 \vee t_3 \vee \neg x_3) \qquad (\neg t_1 \vee \neg x_2)}}{(t_3 \vee \neg x_3 \vee \neg x_2) \qquad (\neg t_3 \vee x_1)}}{(\neg x_3 \vee \neg x_2 \vee x_1)}$$

Conflict analysis returns the clause $(\neg x_3 \vee \neg x_2 \vee x_1)$, which, after being added to P, blocks the model $\neg x_1\, x_2\, x_3$, which does not overlap with the previously found model x_1.

20.7 PROJECTED MODEL ENUMERATION WITHOUT REPETITION

We are given a propositional formula $F(X, Y)$ over the set of irredundant variables X and the set of redundant variables Y, and our task is to enumerate its models projected onto the variables in X. Let $P(X, Y, S)$ and $N(X, Y, T)$ be a dual representation of F according to Section 20.3. Obviously, Equation 20.2–Equation 20.6 hold.

In Figure 20.1, we consider the case with permanent learning of the blocking clauses. Let the first SAT solver execute standard CDCL on P and let I denote its trail. Obviously it finds only total models of P. Due to Equation 20.2, these models satisfy F, too. Now assume a (total) model I of F is found. A second SAT solver SAT is incrementally invoked on $\pi(I, X \cup Y) \wedge N$ with the aim to shrink I obtaining $I^\star$ as described in Section 20.5.

Let b denote the decision level of $\pi(\neg I^\star, X)$ and ℓ be the literal in $\pi(\neg I^\star, X)$ with decision level b. If we now add the clause $\pi(\neg I^\star, X)$ to P and backtrack to decision level $b - 1$, it becomes unit and in the next step $\neg \ell$ is propagated. Notice that $\pi(\neg I^\star, X)$ acts in P as a blocking clause and must not be deleted anytime which might blow up P and slow down the first SAT solver. Moreover, the dual encoding of the blocking clause according to Section 20.6 ensures Equation 20.4, on which our method relies.

In Section 20.7.1, we present the main function EnumerateIrredundant. Unit propagation (Section 20.7.2) and the schema for conflict analysis (Section 20.7.3) are the same as in CDCL for SAT.

20.7.1 *Main Algorithm*

The function EnumerateIrredundant in Figure 20.1 describes the main algorithm. (Black rows 1–18 and 27–28 represent standard CDCL returning a model if the formula under consideration is satisfiable and the empty clause otherwise, blue rows 19–26 the rest of the algorithm.)

Initially, the trail I is empty, the target DNF M is 0, and all variables are unassigned, i. e., assigned decision level ∞. Unit propagation is executed until either a conflict occurs or all variables are assigned a value (line 7).

If a conflict occurs at decision level zero, the search space has been processed exhaustively, and the enumeration terminates (lines 8–11). If a conflict occurs at a decision level higher than zero, conflict analysis is executed (line 13).

If no conflict occurs and all variables are assigned, a total model has been found (line 15). If no relevant decisions are left on the trail I, the search space has been processed exhaustively, the found model is output and the search termi-

Input: formulae $P(X, Y, S)$ and $N(X, Y, T)$ s. t.
$$\exists S\,[\,P(X, Y, S)\,] \equiv \neg\exists T\,[\,N(X, Y, T)\,]\,,$$
set of variables $X \cup Y \cup S \cup T$,
trails I and J

Output: DNF representation of $\pi(P, X)$

```
EnumerateIrredundant ( $P$, $N$ )          // $P_0$ = CNF ( $F$ )
                                            // $N_0 = t_0 \wedge$ CNF ( $t_0 \leftrightarrow \neg F$ )
$I := \varepsilon$
$\delta[\,V \mapsto \infty\,]$
$M := 0$
$i := 0$
forever do
  $i := i + 1$
  $C :=$ PropagateUnits ( $P$, $I$, $\delta$ )
  if $C \neq 0$ then
    $c := \delta(C)$
    if $c = 0$ then
      return $M$
    else
      AnalyzeConflict ( $P$, $I$, $C$, $\delta$ )
  else
    if all variables in $X \cup Y \cup S$ are assigned then
      // $I$ is total model of $P$ and $F$
      if var(decs($I$)) $\cap X = \emptyset$ then
        return $M \vee \pi(I, X)$
      else
        $I^\star :=$ SAT ( $N$, $\pi(I, X \cup Y)$ )
        // $I^\star$ is model of $\pi(F, X \cup Y)$ and conflict set of $I$ w. r. t. $N$
        $P := P \wedge \neg\pi(\text{decs}(I^\star), X)$
        $N := N \setminus \{(t_0 \vee \bigvee_{j=1}^{i-1} t_j)\} \wedge$
                  $(t_0 \vee \bigvee_{j=1}^{i} t_j) \wedge$ CNF ( $t_i \leftrightarrow \pi(\text{decs}(I^\star), X)$ )
        $M := M \vee \pi(I^\star, X)$
        $b := \delta(\neg\pi(I^\star, X))$
        Backtrack ( $I$, $b - 1$ )
    else
      Decide ( $I$ )
```

Figure 20.1: Irredundant model enumeration. The black lines describe CDCL returning a model if one is found and the empty clause otherwise. The blue part represents the extension to model enumeration. A second SAT solver is called incrementally on N assuming the literals on $\pi(I, X \cup Y)$. A conflict occurs by unit propagation only, and $\pi(I^\star, X \cup Y)$ is a (partial) model of F. The is encoded as a dual blocking clause, and P and N are updated accordingly.

```
PropagateUnits ( $F$, $I$, $\delta$ )
1 while some $C \in F$ is unit $(\ell)$ under $I$ do
2   $I := I\ell$
3   $\delta(\ell) := \delta(I)$
4   for all clauses $D \in F$ containing $\neg\ell$ do
5     if $I(D) = 0$ then return $D$
6 return 0
```

```
AnalyzeConflict ( $F$, $I$, $C$, $\delta$ )
1   $D :=$ Learn ( $I$, $C$ )
2   $F := F \wedge D$
3   $\ell :=$ literal in $D$ at decision level $\delta(I)$
4   $j := \delta(D \setminus \{\ell\})$
5 for all literals $k \in I$ with decision level $> j$ do
6   assign $k$ decision level $\infty$
7   remove $k$ from $I$
8 $I := I\ell$
9 $\delta(\ell) := j$
```

Figure 20.2: The function PropagateUnits implements unit propagation in F. The unit literal ℓ is assigned the decision level of I. If some clause $D \in F$ containing the complement of ℓ becomes falsified, PropagateUnits returns D. Otherwise it returns the empty clause 0 indicating that no conflict has occurred. The function AnalyzeConflict is called whenever a clause $C \in F$ becomes empty under the current assignment. It learns a clause D starting with the conflicting clause C. The solver then backtracks to the second highest decision level j in D, upon which D becomes unit with unit literal ℓ, and propagates ℓ.

nates (lines 17–18). If I contains a relevant decision, the found model is shrunken (line 20) by means of dual reasoning as described in Section 20.5. It is blocked, and the last relevant decision literal is flipped (lines 22–26). If no conflict occurs and not all variables are assigned a value, a decision is taken (line 28).

20.7.2 *Unit Propagation*

Unit propagation is described by the function PropagateUnits in Figure 20.2. It takes as input the formula F, the trail I, and the decision level function δ. If a clause $C \in F$ is unit under I, its unit literal ℓ is propagated, i. e., I is extended by ℓ (line 2). Propagated literals are assigned at the current decision level (line 3) as is usual in modern CDCL-based SAT solvers. If the resulting trail falsifies some clause $D \in F$, this clause is returned (lines 4–5). Otherwise the function returns the empty clause 0 (line 6).

20.7.3 *Conflict Analysis*

Conflict analysis is described by the function AnalyzeConflict in Figure 20.2. It takes as input the formula F, the trail I, the conflicting clause C, and the decision level function δ. A clause D is learned as described in Section 20.3.2 and added to F (lines 1–2). The second highest decision level j in D is determined (lines 3–4), and the enumerator backtracks (non-chronologically) to decision level j. Backtracking involves unassigning all literals with decision level higher than j (lines 5–7). After backtracking, the clause D becomes unit with unit literal ℓ, which is propagated and assigned decision level j (lines 8–9).

20.8 FORMALIZING PROJECTED IRREDUNDANT MODEL ENUMERATION

In this section, we provide a formalization of our algorithm presented in Section 20.7. Let $F(X, Y)$ be a formula defined onto the set of relevant (input) variables X and the set of irrelevant (input) variables Y, and assume our task is to enumerate its models projected onto X.

Our formalization works on a dual representation of F, given by $P(X, Y, S)$ and $N(X, Y, T)$, as introduced in Section 20.3.5. So, $P(X, Y, S)$ and $N(X, Y, T)$ are defined over the same sets of relevant variables X and irrelevant variables Y as well as the disjoint sets of variables S and T, respectively, which are defined in terms of the variables in $X \cup Y$. Recall that Equation 20.2–Equation 20.6 hold. We show the working of our calculus by means of an example, before we provide a correctness proof.

20.8.1 *Calculus*

We formalize the algorithm presented in Section 20.7 as a state transition system with transition relation $\leadsto_{\mathsf{EnumIrred}}$. Non-terminal states are described by tuples $(P,\ N,\ M,\ I,\ \delta)$. The third element, M, is a DSOP formula over variables in X. The fourth element, I, denotes the trail defined over variables in $X \cup Y \cup S \cup T$, and δ denotes the decision level function. The initial state is $(P_0,\ N_0,\ 0,\ \varepsilon,\ \delta_0)$, where P_0 and N_0 denote the initial CNF representations of F and $\neg F$, respectively, ε denotes the empty trail, and $\delta_0 \equiv \infty$. The terminal state is given by a DSOP formula M, which is equivalent to the projection of P onto X. The transition relation $\leadsto_{\mathsf{EnumIrred}}$ is the union of transition relations $\leadsto_{\mathsf{R}}$, where R is either End1, End0, Unit, Back1, Back0, DecX, or DecYS. The rules are listed in Figure 20.3.

End1. All variables are assigned and no conflict in P occurred, hence the trail I is a total model of P. It contains no relevant decision indicating that the relevant search space has been processed exhaustively. The model projected onto X is added to M, and the search terminates. It is sufficient to check for relevant decisions, since flipping an irrelevant one would result in detecting redundant models projected onto X. However, due to the addition of blocking clauses, a conflict would occur, and checking for relevant decisions essentially saves work.

End0. A conflict at decision level zero has occurred indicating that the search space has been processed exhaustively. The search terminates leaving M unaltered. We need to make sure no decision is left on the trail, which in particular includes the irrelevant ones. The reason is that after flipping any decision—in particular

End1: $(P, N, M, I, \delta) \rightsquigarrow_{\mathsf{End1}} M \vee m$ if $P|_I \neq 0$ and $(X \cup Y \cup S) - I = \emptyset$ and $\mathsf{var}(\mathsf{decs}(I)) \cap X = \emptyset$ and $m \overset{\text{def}}{=} \pi(I, X)$

End0: $(P, N, M, I, \delta) \rightsquigarrow_{\mathsf{End0}} M$ if exists $C \in P$ with $C|_I = 0$ and $\delta(C) = 0$

Unit: $(P, N, M, I, \delta) \rightsquigarrow_{\mathsf{Unit}} (P, N, M, I\ell^C, \delta[\ell \mapsto a])$ if $P|_I \neq 0$ and exists $C \in P$ with $\{\ell\} = C|_I$ and $a \overset{\text{def}}{=} \delta(I)$

Back1: $(P, N, M, I, \delta) \rightsquigarrow_{\mathsf{Back1}} (P \wedge B, O, M \vee m, J\ell^B, \delta[K \mapsto \infty][\ell \mapsto b])$ if $(X \cup Y \cup S) - I = \emptyset$ and exists $I^\star \leqslant \pi(I, X \cup Y)$ with $JK = I$ such that $N \wedge I^\star \vdash_1 0$ and $m \overset{\text{def}}{=} \pi(I^\star, X)$ and $B \overset{\text{def}}{=} \neg\mathsf{decs}(m)$ and $b + 1 \overset{\text{def}}{=} \delta(B) = \delta(m)$ and $\ell \in B$ and $\ell|_K = 0$ and $b = \delta(B \setminus \{\ell\}) = \delta(J)$ and $O = \mathsf{tseitin}(N \vee \neg B)$

Back0: $(P, N, M, I, \delta) \rightsquigarrow_{\mathsf{Back0}} (P \wedge D^r, N, M, J\ell^D, \delta[K \mapsto \infty][\ell \mapsto j])$ if exists $C \in P$ and exists D with $JK = I$ and $C|_I = 0$ and $\delta(C) = \delta(D) > 0$ such that $\ell \in D$ and $\neg\ell \in \mathsf{decs}(I)$ and $\neg\ell|_K = 0$ and $P \models D$ and $j \overset{\text{def}}{=} \delta(D \setminus \{\ell\}) = \delta(J)$

DecX: $(P, N, M, I, \delta) \rightsquigarrow_{\mathsf{DecX}} (P, N, M, I\ell^d, \delta[\ell \mapsto d])$ if $P|_I \neq 0$ and $\mathsf{units}(P|_I) = \emptyset$ and $\delta(\ell) = \infty$ and $d \overset{\text{def}}{=} \delta(I) + 1$ and $\mathsf{var}(\ell) \in X$

DecYS: $(P, N, M, I, \delta) \rightsquigarrow_{\mathsf{DecYS}} (P, N, M, I\ell^d, \delta[\ell \mapsto d])$ if $P|_I = 0$ and $\mathsf{units}(P|_I) = \emptyset$ and $\delta(\ell) = \infty$ and $d \overset{\text{def}}{=} \delta(I) + 1$ and $\mathsf{var}(\ell) \in Y \cup S$ and $X - I = \emptyset$

Figure 20.3: Rules for projected model enumeration without repetition. States are represented as tupels (P, N, M, I, δ). The formulae $P(X, Y, S)$ and $N(X, Y, T)$ are a dual representation of the formula $F(X.Y)$, whose models projected onto X are sought. These models are recorded in the initially empty DNF M. The last two elements, I and δ, denote the current trail and decision level function, respectively. If a model is found or a conflict encountered and the search space has been exhaustively processed, the search terminates (rules End1 and End0). Otherwise, the model is shrunken and a dual blocking clause added (rules Back1) or conflict analysis executed followed by non-chronological backtracking (rule Back0). If the residual of P under the current trail IJ contains a unit literal, this is propagated (rule Unit). If none of the mentioned preconditions are met, a decision is taken. Relevant literals are prioritized (rule DecX) over irrelevant and internal ones (rule DecYS).

also irrelevant and internal ones—the resulting trail might be extended to a model of P.

Unit. No conflict in P occurred, and a clause in P is unit under I. Its unit literal ℓ is propagated and assigned the current decision level.

Back1. All variables are assigned and no conflict in P occurred, hence the trail I is a total model of P. It is shrunken as described in Section 20.5 obtaining $I^\star$. The projection of $I^\star$ onto X, m, is added to M. The clause B consisting of the negated decision literals of m is added as a blocking clause to P. Its negation $\neg B$ is added disjunctively to N, which is transformed back into CNF by means of the Tseitin transformation. The solver backtracks to the second highest decision level in B and propagates ℓ at the current decision level, i. e., basically flips the relevant decision literal with highest decision level.

Back0. The current trail falsifies a clause in P at a decision level greater than zero indicating that the search space has not yet been processed exhaustively. Conflict analysis returns a clause D implied by P, which is added to P and marked as redundant. The solver backtracks to the second highest decision level j in D. The learned clause D becomes unit, and its unit literal ℓ is propagated at decision level j. In contrast to End1, any decision literal need be flipped, which particularly applies to irrelevant and internal decision literals.

DecX. No conflict has occurred, the residual of P under I contains no units, and there is an unassigned relevant literal ℓ. The current decision level is incremented to d, ℓ is decided and assigned decision level d.

DecYS. No conflict has occurred, and the residual of P under I contains no units. All relevant literals are assigned, and there is an unassigned irrelevant or internal literal ℓ. The current decision level is incremented to d, ℓ is decided and assigned decision level d.

20.8.2 *Example*

The working of our calculus is shown by means of an example. Consider again Example 20.1 and Example 20.5. We have

$$F = \underbrace{(a \vee c)}_{C_1} \wedge \underbrace{(a \vee \neg c)}_{C_2} \wedge \underbrace{(b \vee d)}_{C_3} \wedge \underbrace{(b \vee \neg d)}_{C_4}$$

and assume the set of relevant variables is $X = \{a, c\}$ and the set of irrelevant variables is $Y = \{b, d\}$. The formula F is already in CNF, therefore we define $P_0 = F$ and accordingly $S_0 = \emptyset$. For its negation

$$\neg F = (\neg a \wedge \neg c) \vee (\neg a \wedge c) \vee (\neg b \wedge \neg d) \vee (\neg b \wedge d)$$

we define

$$\begin{aligned} N_0 = {} & \underbrace{(\neg t_1 \vee \neg a)}_{D_1} \wedge \underbrace{(\neg t_1 \vee \neg c)}_{D_2} \wedge \underbrace{(a \vee c \vee t_1)}_{D_3} \wedge \\ & \underbrace{(\neg t_2 \vee \neg a)}_{D_4} \wedge \underbrace{(\neg t_2 \vee c)}_{D_5} \wedge \underbrace{(a \vee \neg c \vee t_2)}_{D_6} \wedge \\ & \underbrace{(\neg t_3 \vee \neg b)}_{D_7} \wedge \underbrace{(\neg t_3 \vee \neg d)}_{D_8} \wedge \underbrace{(b \vee d \vee t_3)}_{D_9} \wedge \\ & \underbrace{(\neg t_4 \vee \neg b)}_{D_{10}} \wedge \underbrace{(\neg t_4 \vee d)}_{D_{11}} \wedge \underbrace{(b \vee \neg d \vee t_4)}_{D_{12}} \wedge \\ & \underbrace{(t_1 \vee t_2 \vee t_3 \vee t_4)}_{D_{13}} \end{aligned}$$

Table 20.1: Execution trace for $F = (a \vee c) \wedge (a \vee \neg c) \wedge (b \vee d) \wedge (b \vee \neg d)$ defined over the set of relevant variables $X = \{a, c\}$ and the set of irrelevant variables $Y = \{b, d\}$ (see also Example 20.1 and Example 20.5).

STEP	RULE	I	$P\|_I$	N	M
0		ε	P_0	N_0	0
1	DecX	a^d	$(b \vee d) \wedge (b \vee \neg d)$	N_0	0
2	DecX	$a^d\, c^d$	$(b \vee d) \wedge (b \vee \neg d)$	N_0	0
3	DecYS	$a^d\, c^d\, b^d$	1	N_0	0
4	DecYS	$a^d\, c^d\, b^d\, d^d$	1	N_0	0
5	Back1	$\neg a^{B_1}$	$(c) \wedge (\neg c) \wedge (b \vee d) \wedge (b \vee \neg d)$	N_1	a
6	Unit	$\neg a^{B_1}\, c^{C_1}$	$() \wedge (b \vee d) \wedge (b \vee \neg b)$	N_1	a
7	End0				a

with the set of internal variables $T_0 = \{t_1, t_2, t_3, t_4\}$. Assume a lexicographic ordering of the input variables, i. e., $a \succ_{lex} b \succ_{lex} c \succ_{lex} d$, and assume we choose the decision variable according to this ordering. The execution steps are depicted in Table 20.1.

Step 0: The initial state is given by the empty trail ε, the CNF formulae P_0 and N_0, and the empty DNF formula 0.

Step 1: The formula P_0 contains no units, and there are unassigned relevant variables. The preconditions of rule DecX are met, and decision a is taken.

Step 2: No conflict occurred, $P_0|_I$ contains no units, and there are unassigned relevant variables. The preconditions of rule DecX are met, and c is decided.

Step 3: No conflict occurred, $P_0|_I$ contains no units. All relevant variables are assigned, and there are unassigned irrelevant variables. The preconditions of rule DecYS are met, and decision b is taken. Notice that I already satisfies P_0, but the solver is not able to detect this fact.

Step 4: Again, the preconditions of rule DecYS are met, and decision d is taken.

Step 5: No conflict occurred and all variables are assigned, hence I is a model of P_0. It is shrunken following the procedure described in Section 20.5.

We call a SAT solver incrementally on $N_0 \wedge I$, i. e., assuming the literals on I. A conflict in N_0 occurs by propagation of variables in T_0 only, and conflict analysis provides us with the shrunken model $a\, b$ of F. Below, the resulting implication graph and trail are depicted:

$$\begin{tikzcd}
 & \neg t_1 \arrow[ddr, "D_{13}"] & \\
a \arrow[ur, bend left, "D_1"] \arrow[r, "D_4"] & \neg t_2 \arrow[dr, "D_{13}"] & \\
c & & \kappa \\
b \arrow[r, "D_7"'] \arrow[dr, bend right, "D_{11}"'] & \neg t_3 \arrow[ur, "D_{13}"'] & \\
d & \neg t_4 \arrow[uur, "D_{13}"'] &
\end{tikzcd}$$

$$I = a^a\, c^a\, b^a\, d^a\, \neg {t_1}^{D_1}\, \neg {t_2}^{D_4}\, \neg {t_3}^{D_7}\, \neg {t_4}^{D_{10}}$$

The conflicting clause is D_{13}. For conflict analysis, we resolve D_{13} with D_{10}, the resolvent with D_7, followed by resolution with D_4 and D_1. The obtained clause $(\neg b \lor \neg a)$ contains only assumed literals. The assumptions c and d do not participate in the conflict and therefore do not occur in the resulting clause. Below, the resolution steps are visualized.

$$\cfrac{\cfrac{\cfrac{\cfrac{(t_1 \lor t_2 \lor t_3 \lor t_4) \qquad (\neg t_4 \lor \neg b)}{(t_1 \lor t_2 \lor t_3 \lor \neg b)} \qquad (\neg t_3 \lor \neg b)}{(t_1 \lor t_2 \lor \neg b)} \qquad (\neg t_2 \lor \neg a)}{(t_1 \lor \neg b \lor \neg a)} \qquad (\neg t_1 \lor \neg a)}{(\neg b \lor \neg a)}$$

The negation of $(\neg b \lor \neg a)$ is $a \land b$, hence $I^\star = a\,b \leqslant I$. The first projected model is $m_1 = \pi(I^\star, X) = a$ and accordingly $M_1 = M_0 \lor m_1$. Furthermore, we have $B_1 = \neg\mathsf{decs}(m_1) = (\neg a)$, and

$$\begin{aligned}
P_1 &= P_0 \land \underbrace{(\neg a)}_{B_1} \quad \text{and} \\
N_1 &= N_0 \lor \underbrace{(a)}_{\neg B_1} \\
&= N_0 \setminus \{(\bigvee_{j=1}^{4} t_j)\} \land (\bigvee_{j=1}^{5} t_j) \land (t_5 \leftrightarrow a) \\
&= (\bigwedge_{i=1}^{12} D_i) \land \underbrace{(\neg t_5 \lor a)}_{D_{14}} \land \underbrace{(\neg a \lor t_5)}_{D_{15}} \land \underbrace{(t_1 \lor t_2 \lor t_3 \lor t_4 \lor t_5)}_{D_{16}}
\end{aligned}$$

where $D_{14} \land D_{15} = (t_5 \leftrightarrow a)$ is the Tseitin transformation of m_1. The clause $\neg B_1$ is added disjunctively to N_0. To retain N in CNF, $\neg B_1$ is encoded as $(t_5 \leftrightarrow \neg B_1)$, t_5 is added to D_{13} resulting in D_{16}, and $T_1 = T_0 \cup \{t_5\} = \{t_1, t_2, t_3, t_4, t_5\}$ as described in Section 20.6. The clause B_1 acts in P as blocking clause. The solver backtracks to decision level zero and propagates $\neg a$ with reason B_1.

Step 6: The formula $P_1|_I$ contains two units, $C_1|_I = (c)$ and $C_2|_I = (\neg c)$. The literal c is propagated with reason C_1.

Step 7: The trail falsifies C_2, and the current decision level is zero. The preconditions of rule End0 are met and the search terminates without altering $M = a$, which represents exactly the models of F projected onto X, namely $a\,c$ and $a\,\neg c$.

20.8.3 *Proofs*

Our proofs are based on the ones provided for our work addressing chronological CDCL for model counting [139], which in turn rely on the proof of correctness we provided for chronological CDCL [138]. The method presented here mainly differs from the former in the following aspects: The total models found are shrunken by means of dual reasoning. It adopts non-chronological CDCL instead of chronological CDCL and accordingly makes use of blocking clauses, which affects the ordering of the literals on the trail. In fact, unlike in chronological CDCL, the literals on the trail are ordered in ascending order with respect to their decision level,

InvDualPN:	$\exists S\,[\,P(X,Y,S)\,] \equiv \neg\exists T\,[\,N(X,Y,T)\,]$
InvDecs:	$\delta(\mathsf{decs}(I)) = \{1,\ldots,\delta(I)\}$
InvImplIrred:	$\forall n \in \mathbb{N}\,.\,P \wedge \mathsf{decs}_{\leqslant n}(I) \models I_{\leqslant n}$
InvDSOP:	M is a DSOP

Figure 20.4: Invariants for projected model enumeration without repetition.

which simplifies not only the rules but also the proofs. Projection in turn adds complexity to some invariants. In some aspects our proofs are similar to or essentially the same as those in our former proofs [138, 139]. However, they are fully worked out to keep them self-contained.

In order to prove the correctness of our method, we make use of the invariants listed in Figure 20.4. Invariant InvDualPN in essence is Equation 20.4. It ensures that the shrunken model is again a model of P projected onto the input variables stating that P and N projected onto the input variables $X \cup Y$ are each other's negation. Intuitively, Invariant InvDualPN holds because the found models are blocked in P and added to its negation N. Invariants InvDecs and InvImplIrred equal Invariants (2) and (3) in our proofs of correctness of chronological CDCL [138] and model counting by means of chronological CDCL [139]. Invariant InvImplIrred differs from the latter in that we need not consider the negation of the DNF M explicitly. The negation of M is exactly the conjunction of the blocking clauses associated with the found models, and these are added to P. Invariant InvImplIrred is needed to show that the literal propagated after backtracking is implied by the resulting trail. Its reason is either a blocking clause (rule Back1) or a clause learned by means of conflict analysis (rule Back0).

Our proof is split into several parts. We start by showing that the invariants listed in Figure 20.4 hold in non-terminal states (Section 20.8.3.1). Then we prove that our method always makes progress (Section 20.8.3.2), before showing that our procedure terminates (Section 20.8.3.3). We conclude the proof by showing that every total model is found exactly once and that all total models are detected, i. e., that upon termination $M \equiv \pi(P,X)$ holds (Section 20.8.3.4).

20.8.3.1 *Invariants in Non-Terminal States*

Proposition 20.2. *Invariants InvDualPN, InvDecs, InvDSOP, and InvImplIrred hold in non-terminal states.*

Proof. The proof is carried out by induction over the number of rule applications. Assuming Invariants InvDualPN to InvImplIrred hold in a non-terminal state $(P,\ N,\ M,\ I,\ \delta)$, we show that they are met after the transition to another non-terminal state for all rules.

Unit

Invariant InvDualPN: Neither P nor N are altered, hence Invariant InvDualPN holds after the application of rule Unit.

Invariant InvDecs: The trail I is extended by a literal ℓ. We need to show that ℓ is not a decision literal. Only the case where $a > 0$ need be considered, since at decision level zero all literals are propagated. There exists a clause $C \in P$ s. t.$C|_I =$

$\{\ell\}$. Now, $a = \delta(I)$, i. e., there is already a literal $k \neq \ell$ on I with $\delta(k) = a$. From this it follows that ℓ is not a decision literal. The decisions remain unchanged, and Invariant InvDecs holds after applying rule Unit.

Invariant InvImplIrred: Due to $C|_I = \{\ell\}$, we have $P \wedge \mathsf{decs}_{\leqslant n}(I) \models \neg(C \setminus \{\ell\})$. Since $C \in P$, also $P \wedge \mathsf{decs}_{\leqslant n}(I) \models C$. Modus ponens gives us $P \wedge \mathsf{decs}_{\leqslant n}(I) \models I_{\leqslant n}$. Hence, $P \wedge \mathsf{decs}_{\leqslant n}(I\ell) \models I\ell_{\leqslant n}$, and Invariant InvImplIrred holds after executing rule Unit.

Invariant InvDSOP: Due to the premise, M is a DSOP. It is not altered by rule Unit and after its application is therefore still a DSOP.

Back1

Invariant InvDualPN: We have $\exists\, S\, [\, P(X,Y,S)\,] \equiv \neg \exists\, T\, [\, N(X,Y,T)\,]$ and we need to show $\exists\, S\, [\, (P \wedge B)(X,Y,S)\,] \equiv \neg\exists\, T\, [\, O(X,Y,T)\,]$ where $B = \neg\mathsf{decs}(m)$ and $B = \mathsf{tseitin}(N \vee \neg B)$ and $m = \pi(I^\star, X)$ is a model of P projected onto X. Since we have that $\exists\, T\, [\, O(X,Y,T)\,] \equiv \exists\, T\, [\, (N \vee \neg B)(X,Y,T)\,]$, and $\neg\exists\, T\, [\, (N \vee \neg B)(X,Y,T)\,] \equiv \forall\, T\, [\, (\neg N \wedge B)(X,Y,T)\,]$, we reformulate the claim as $\exists\, S\, [\, (P \wedge B)(X,Y,S)\,] \equiv \forall\, T\, [(\neg N \wedge B)(X,Y,T)\,]$. Together with $\exists\, S\, [\, P(X,Y,S)\,] \equiv \forall\, T\, [\, \neg[N(X,Y,T)\,]$ and observing that B contains no variable in Y, the claim holds.

Invariant InvDecs: We show that the decisions remaining on the trail are unaffected and that no new decision is taken, i. e., ℓ in the post state is not a decision. It is sufficient to consider the case where $\delta(I) > 0$. Now, $J = I_{\leqslant b}$ by the definition of J, and the decisions on J are not affected by rule Back1. We have $\delta(B \setminus \{\ell\}) = b = \delta(J)$ and $\delta(B) = b + 1$. Since relevant decisions are prioritized, also $B = \neg\mathsf{decs}_{\leqslant b+1}(\pi(I,X)) = \neg\mathsf{decs}_{\leqslant b+1}(I)$. According to Invariant InvDecs there exists exactly one decision literal for each decision level and in particular in B. Since $\ell \in B$, we have $\neg\ell \in \mathsf{decs}(I)$. Precisely, $\neg\ell \in K$, and $\neg\ell$ is unassigned upon backtracking. Due to the definition of B there exists a literal $k \in B$ where $k \neq \ell$ such that $\delta(k) = b$, i. e., $k \in J$, hence k precedes ℓ on the resulting trail. By the definition of the blocks on the trail, ℓ is not a decision literal. Since the decisions on J are unaffected, as argued above, Invariant InvDecs is met.

Invariant InvImplIrred: We need to show that $P \wedge \mathsf{decs}_{\leqslant n}(J\ell) \models (J\ell)_{\leqslant n}$ for all n. First notice that the decision levels of the literals in J do not change by applying rule Back1. Only the decision level of the variable of ℓ is decremented from $b + 1$ to b. It also stops being a decision. Since $\delta(J\ell) = b$, we can assume $n \leqslant b$. Observe that $P \wedge \mathsf{decs}_{\leqslant n}(J\ell) \equiv P \wedge \mathsf{decs}_{\leqslant n}(J)$, since ℓ is not a decision in $J\ell$ and $I_{\leqslant b} = J$ and thus $I_{\leqslant n} = J_{\leqslant n}$ by definition. Now the induction hypothesis is applied and we get $P \wedge \mathsf{decs}_{\leqslant n}(J\ell) \models I_{\leqslant n}$. Again using $I_{\leqslant n} = J_{\leqslant n}$ this almost closes the proof except that we are left to prove $P \wedge \mathsf{decs}_{\leqslant b}(J\ell) \models \ell$ as ℓ has decision level b in $J\ell$ after applying the rule and thus ℓ disappears in the proof obligation for $n < b$. To see this notice that $P \wedge \neg B \models I_{\leqslant b+1}$ using again the induction hypothesis for $n = b + 1$, and recalling that relevant decisions are prioritized, i. e., $I_{\leqslant b+1}$ contains only relevant decisions, and $\neg B = \mathsf{decs}(\pi(I^\star, X)) = \mathsf{decs}_{\leqslant b+1}(I)$. This gives $P \wedge \neg\mathsf{decs}_{\leqslant b}(J) \wedge \neg\ell \models I_{\leqslant b+1}$ and thus $P \wedge \neg\mathsf{decs}_{\leqslant b}(J) \wedge \neg I_{\leqslant b+1} \models \ell$ by conditional contraposition. Therefore, Invariant InvImplIrred holds.

Invariant InvDSOP: We assume that M is a DSOP and need to show that $M \vee m$ is also a DSOP. Due to the use of the dual blocking clause encoding, Proposition 20.1 holds, and invariant InvDSOP is met after executing Back1.

Back0

Invariant InvDualPN: We have $\exists\, S\, [\, P(X,Y,S)\,] \equiv \neg\exists\, T\, [\, N(X,Y,T)\,]$, and we need to show that $\exists\, S\, [\, (P \wedge D)(X,Y,S)\,] \equiv \neg\exists\, T\, [\, N(X,Y,T)\,]$. By the premise, $P \models$

D, and hence $P \wedge D \equiv P$. Now $\exists S\,[(P \wedge D)(X,Y,S)] \equiv \exists S\,[P(X,Y,S)] \equiv \neg\exists T\,[N(X,Y,T)]$, and Invariant InvDualPN holds.

Invariant InvDecs: We have $J \leqslant I$, hence the decisions on J remain unaltered. Now we show that ℓ is not a decision literal. As in the proof for rule Unit, it is sufficient to consider the case where $j > 0$. There exists a clause D where $P \models D$ such that $\delta(D) > 0$ and a literal $\ell \in D$ for which $\ell|_K = 0$ and $\neg\ell \in K$, hence ℓ is unassigned during backtracking. Furthermore, there exists a literal $k \in D$ where $k \neq \ell$ and such that $\delta(k) = j$ which precedes ℓ on the trail $J\ell$. Therefore, following the argument in rule Unit, the literal ℓ is not a decision literal. Since the decisions remain unchanged, Invariant InvDecs holds after applying rule Back0.

Invariant InvImplIrred: Let n be arbitrary but fixed. Before executing rule Back0, we have $P \wedge \mathsf{decs}_{\leqslant n}(I) \models I_{\leqslant n}$. We need to show that $P \wedge \mathsf{decs}_{\leqslant n}(J\ell) \models (J\ell)_{\leqslant n}$. Now, $I = JK$ and $J < I$, i. e., $P \wedge \mathsf{decs}_{\leqslant n}(J) \models J_{\leqslant n}$. From $j = \delta(D \setminus \{\ell\}) = \delta(J)$ we get $D|_J = \{\ell\}$. On the one hand, $P \wedge \mathsf{decs}_{\leqslant n}(J) \models \neg(D \setminus \{\ell\})$, and on the other hand $P \wedge \mathsf{decs}_{\leqslant n}(J) \models D$. Therefore, by modus ponens, $P \wedge \mathsf{decs}_{\leqslant n}(J) \models \ell$. Since ℓ is not a decision literal, as shown above, $P \wedge \mathsf{decs}_{\leqslant n}(J) \equiv P \wedge \mathsf{decs}_{\leqslant n}(J\ell)$ and $P \wedge \mathsf{decs}_{\leqslant n}(I\ell) \models I\ell$, and Invariant InvImplIrred holds after applying rule Back0.

Invariant InvDSOP: The DSOP M remains unaltered, and InvDSOP still holds after executing rule Back0.

DecX

Invariant InvDualPN: Both P and N remain unaltered, hence Invariant InvDualPN still holds after executing rule DecX.

Invariant InvDecs: The literal ℓ is a decision literal by definition. It is assigned decision level $d = \delta(I) + 1$. Since $\ell \in \mathsf{decs}(I\ell)$, we have $\delta(\mathsf{decs}(I\ell)) = \{1, \ldots, d\}$, and Invariant InvDecs holds after applying rule DecX.

Invariant InvImplIrred: Le n be arbitrary but fixed. Since ℓ is a decision literal, we have $P \wedge \mathsf{decs}_{\leqslant n}(I\ell) \equiv P \wedge \mathsf{decs}_{\leqslant n}(I) \wedge \ell \models I_{\leqslant n} \wedge \ell \equiv (I\ell_{\leqslant n}$. Hence, Invariant InvImplIrred holds after applying rule DecX.

Invariant InvDSOP: The DSOP M remains unaltered by rule DecX, hence invariant InvDSOP still holds after its application.

DecYS.

The proofs of Invariants InvDualPN, InvDecs, InvDSOP, and InvDSOP are identical to the ones for rule DecX. □

20.8.3.2 *Progress*

Our method can not get caught in an endless loop, as shown next.

Proposition 20.3. *EnumerateIrredundant always makes progress, i. e., in every non-terminal state a rule is applicable.*

Proof. The proof is executed by induction over the number of rule applications. We show that in any non-terminal state $(P,\ N,\ M,\ I,\ \delta)$ a rule is applicable.

Assume all variables are assigned and no conflict has occurred. If no relevant decision is left on the trail I, rule End1 can be applied. Otherwise, we execute an incremental SAT call $\mathsf{SAT}(N, \pi(I, X \cup Y))$. Since all input variables are assigned, we obtain a conflict by propagating internal variables only. Conflict analysis gives us the subsequence $I^\star$ of $\pi(I, X \cup Y)$ consisting of the literals involved in the conflict, which is a model of F. Since we are interested in the models of F projected onto X, we choose $B = \neg\mathsf{decs}(\pi(I^\star, X))$. Now, $\delta(B) = b + 1$, and due to Invariant InvDecs B contains exactly one decision literal ℓ such that $\delta(\ell) = b + 1$ and therefore $\delta(B \setminus \{\ell\}) = b$. We choose J and K such that $I = JK$ and $b = \delta(J)$ and in particular $\ell|_K = 0$. After backtracking to decision level b, we have $I_{\leqslant b} = J$ where $B|_J = \{\ell\}$. All preconditions of rule Back1 are met.

If instead a conflict has occurred, a clause $C \in P$ exists such that $C|_I = 0$. If $\delta(C) = 0$, rule End0 is applicable. Otherwise, by Invariant InvImplIrred we have $P \wedge \mathsf{decs}_{\leqslant\delta(I)}(I) \equiv P \wedge \mathsf{decs}_{\leqslant\delta(I)}(I) \wedge I_{\leqslant\delta(I)} \models I_{\leqslant\delta(I)}$. Since $I(P) \equiv 0$, also $P \wedge \mathsf{decs}_{\leqslant\delta(I)}(I) \wedge I_{\leqslant\delta(I)} \equiv P \wedge \mathsf{decs}_{\leqslant\delta(I)}(I) \equiv 0$. If we choose $D = \neg\mathsf{decs}(I)$ we obtain $P \wedge \neg D \wedge I_{\leqslant\delta(I)} \equiv 0$, thus $P \models D$. Clause D contains only decision literals and $\delta(D) = \delta(I)$. From Invariant InvDecs we know that D contains exactly one decision literal for each decision level in $\{1, \ldots, \delta(I)\}$. We choose $\ell \in D$ such that $\delta(\ell) = \delta(I)$. Then the asserting level is given by $j = \delta(D \setminus \{\ell\})$. Without loss of generalization we assume the trail to be of the form $I = JK$ where $\delta(J) = j$. After backtracking to decision level j, the trail is equal to J. Since $D|_J = \{\ell\}$, all conditions of rule Back0 hold.

If $P|_I \notin \{0, 1\}$, there are unassigned variables in $X \cup Y \cup S$. If there exists a clause $C \in P$ where $C|_I = \{\ell\}$, the preconditions of rule Unit are met. If instead $\mathsf{units}(F|_I) = \emptyset$, there exists a literal ℓ with $\mathsf{var}(\ell) \in X \cup Y \cup S$ and $\delta(\ell) = \infty$. If not all relevant variables are assigned, the preconditions of rule DecX are satisfied. Otherwise, rule DecYS is applicable.

All possible cases are covered by this argument. Hence, in every non-terminal state a rule is applicable, i. e., EnumerateIrredundant always makes progress. □

20.8.3.3 *Termination*

Proposition 20.4. *EnumerateIrredundant terminates.*

Proof. In our proof we follow the argument by Nieuwenhuis et al. [155] and Marić and Janičić [124], or more precisely the one by Blanchette et al. [29].

We need to show that from the initial state $(P, N, 0, \varepsilon, \delta_0)$ a final state M is reached in a finite number of steps, i. e., no infinite sequence of rule applications is generated. Otherwise stated, we need to prove that the relation $\leadsto_{\mathsf{EnumIrred}}$ is well-founded. To this end, we define a well-founded relation $\succ_{\mathsf{EnumIrred}}$ such that any transition $s \leadsto_{\mathsf{EnumIrred}} s'$ from a state s to a state s' implies $s \succ_{\mathsf{EnumIrred}} s'$.

In accordance with Blanchette et al. [29] but adopting the notation introduced by Fleury [78], we map states to lists. Using the abstract representation of the assignment trail I by Nieuwenhuis et al. [155], we write

$$I = I_0\, \ell_1\, I_1\, \ell_2\, I_2 \,\ldots\, \ell_m\, I_m \quad \text{where} \quad \{\ell_1, \ldots, \ell_m\} = \mathsf{decs}(I). \tag{20.13}$$

The state (P, N, M, I, δ) is then mapped to

$$[\underbrace{0, \ldots, 0}_{|I_0|}, 1, \underbrace{0, \ldots, 0}_{|I_1|}, 1, \underbrace{0, \ldots, 0}_{|I_2|}, 1, \ldots, 1, \underbrace{0, \ldots, 0}_{|I_m|}, \underbrace{2, \ldots, 2}_{|V|-|I|}] \tag{20.14}$$

where $V = X \cup Y \cup S$. In this representation, the order of the literals on I is reflected. Propagated literals are denoted by 0, decisions are denoted by 1. Unassigned literals are represented by 2 and are moved to the end. The final state M is represented by ε. The state containing the trail I in Equation 20.13 is mapped to the list in Equation 20.14. The first $|I_0|$ entries represent the literals propagated at decision level zero, the 1 at position $|I_0| + 1$ represents the decision literal ℓ_1, and so on for all decision levels on I. The last $|V| - |I|$ entries denote

$$\frac{[\ldots]}{\varepsilon}\ \mathsf{End1} \qquad \frac{[0,\ldots,0,2,\ldots,2]}{\varepsilon}\ \mathsf{End0} \qquad \frac{[\ldots,2,2,\ldots,2]}{[\ldots,0,2,\ldots,2]}\ \mathsf{Unit}$$

$$\frac{[\ldots,1,\ldots,1,\ldots,2,\ldots,2]}{[\ldots,0,2,\ldots\ldots\ldots\ldots,2]}\ \mathsf{Back1} \qquad \frac{[\ldots,1,\ldots,1,\ldots,2,\ldots,2]}{[\ldots,0,2,\ldots\ldots\ldots\ldots,2]}\ \mathsf{Back0}$$

$$\frac{[\ldots,2,2,\ldots,2]}{[\ldots,1,2,\ldots,2]}\ \mathsf{DecX} \qquad \frac{[\ldots,2,2,\ldots,2]}{[\ldots,1,2,\ldots,2]}\ \mathsf{DecYS}$$

Figure 20.5: Transitions of states mapped to lists according to Equation 20.14. The initial state is depicted above the horizontal rule, the resulting state below. The two end rules lead to the minimal element ε. Rule Unit replaces an unassigned literal (denoted by 2) by a propagated one (denoted by 0) and leaves the rest unchanged. Rules Back1 and Back0 replace a decision literal (denoted by 1) by a propagated one. Finally, the two decision rules replace an unassigned literal by a decision. Clearly, w. r. t. the lexicographic order, the states decrease by a rule application.

the unassigned variables. Notice that we are not interested in the variable assignment itself, but in its structure, i. e., the number of propagated literals per decision level and the number of unassigned variables. Furthermore, the states are encoded into lists of the same length. This representation induces a lexicographic order $>_{\mathsf{lex}}$ on the states. We therefore define $\succ_{\mathsf{EnumIrred}}$ as the restriction of $>_{\mathsf{lex}}$ to $\{[v_1,\ldots,v_{|V|}] \mid v_i \in \{0,1,2\} \text{ for } 1 \leqslant i \leqslant |V|\}$. Accordingly, we have that $s \succ_{\mathsf{EnumIrred}} s'$, if $s >_{\mathsf{lex}} s'$.

In Figure 20.5, the state transitions for the rules are visualized. In this representation, the unspecified elements occurring prior to the first digit are not altered by the application of the rule. We show that $s \succ_{\mathsf{EnumIrred}} s'$ for each rule.

End1. The state s' is mapped to ε, which is the minimal element with respect to $>_{\mathsf{lex}}$, hence $s \succ_{\mathsf{EnumIrred}} s'$ trivially holds. The representation of the state may contain both 0's and 1's but no 2's, since our algorithm detects only total models.[13] Recall that the associated trail must not contain any relevant decision, which is not reflected in the structure of the trail.

End0. The state s' is mapped to ε, which is the minimal element with respect to $>_{\mathsf{lex}}$, hence $s \succ_{\mathsf{EnumIrred}} s'$ trivially holds. The representation of the state may contain both 0's and 2's but no 1's, since any decision need be flipped.

Unit. An unassigned variable is propagated. Its representation changes from 2 to 0, and all elements preceding it remain unaffected. Due to $2 >_{\mathsf{lex}} 0$, we also have that $s \succ_{\mathsf{EnumIrred}} s'$.

Back1 / Back0. A decision literal, e. g., $\neg\ell$ is flipped and propagated at a lower decision level, let's say d. The decision level d is extended by ℓ, which is represented by 0, and replaces the decision literal at decision level $d+1$. All other literals at decision level $d+1$ and higher are unassigned and thus represented by 2. Therefore, $s \succ_{\mathsf{EnumIrred}} s'$. Notice that, although different preconditions of the rules Back1 and Back0 apply and the two rules differ, the structure of their states is the same.

13 This restriction may be weakened in favor of finding partial models. In this section, we refer to the rules introduced in Section 20.8.1 and discuss a generalization of our algorithm enabling the detection of partial models further down.

DecX / DecYS. An unassigned variable is decided, i. e., in the representation, the first occurrence of 2 is replaced by 1. The other elements remain unaltered, hence $s \succ_{\mathsf{EnumIrred}} s'$. As for the backtracking rules, whether a relevant or irrelevant or internal variable is decided, is irrelevant and not reflected in the mapping of the state, as for rules Back1 and Back0.

We have shown that after any rule application the resulting state is smaller than the preceding one with respect to the lexicographic order on which $\succ_{\mathsf{EnumIrred}}$ is based. This argument shows that $\succ_{\mathsf{EnumIrred}}$ is well-founded, and that therefore EnumerateIrredundant terminates. □

20.8.3.4 *Equivalence*

The final state is given by a DSOP M such that $M \equiv \pi(F, X)$. The proof is split into several steps. We start by proving that, given a total model I of P, its subsequence $I^\star$ returned by SAT (line 20 of EnumerateIrredundant in Figure 20.1) is a (partial) model of $\pi(P, X)$ and that any total model of P found during execution either was already found or is found for the first time. Then we show that all models of P are found and that each model is found exactly once, before concluding by proving that $M \equiv \pi(F, X)$.

Proposition 20.5. *Let I be a total model of P and $I^\star = SAT(N, \pi(I, X \cup Y))$. Then $I^\star$ is a model of $\pi(P, X)$.*

Proof. All variables in $X \cup Y \cup S$ are assigned, and $P(X, Y, S)$ and $N(X, Y, T)$ are a dual representation of $F(X, Y)$. Invariant InvDualPN holds. In particular it holds for the values of the variables in $X \cup Y \cup S$ set to their values in I, i. e., we have that $\exists S\,[\,P(X, Y, S)|_I\,] \equiv \neg\exists T\,[\,N(X, Y, T)|_I\,]$, where only the unassigned variables in $(X \cup Y) - I$ are universally quantified. Since I is a total model of P, Invariant InvDualPN can be rewritten as $P(X, Y, S)|_I \equiv \neg\exists T\,[\,N(X, Y, T)|_I$, and $\pi(I, X \cup Y)$ can not be extended to a model of N. Since the variables in T are defined in terms of variables in $X \cup Y$, an incremental SAT call on $N \wedge I$ yields a conflict in N exclusively by propagating variables in T.

Exhaustive conflict analysis yields a clause D consisting of the negations of the (assumed) literals in I involved in the conflict. Its negation is a countermodel of $\pi(N, X \cup Y)$, which due to Equation 20.5 is a model of $\pi(P, X \cup Y)$. Obviously, the same holds for the projection onto X, and $\pi(\neg D, X) \models \pi(P, X)$. Since $I^\star = \pi(\neg D, X)$, we have $I^\star \models \pi(P, X)$, and the claim holds. □

Note: Proposition 20.5 corresponds to soundness according to Definition 20.4.

Proposition 20.6. *A total model I of P is either*

(i) contained in M or

(ii) subsumed by a model in M or

(iii) a model of $P_0 \wedge \bigwedge_i B_i$ where B_i are the blocking clauses added to P_0

Proof. All variables in $X \cup Y \cup S$ are assigned, and $I \models P$. If I was already found earlier, it was shrunken and the resulting model projected onto X to obtain $I^\star$, which was then added to M (rule Back1 and line 24 in EnumerateIrredundant in Figure 20.1). If all assumed variables participated in the conflict and furthermore

$Y = S = \emptyset$, $\pi(I^\star, X \cup Y) = \pi(I^\star, X) = I$, and (i) holds. Otherwise, $I^\star < I$, and $I^\star$ subsumes I. Since $I^\star \in M$, in this case (ii) holds.

Suppose the model I is found for the first time. Since $I \models P$, also $I \models C$ for all clauses $C \in P$. This in particular holds for all blocking clauses which were added to the original formula $P = P_0$ (rule Back1 and line 22 in EnumerateIrredundant), and (iii) holds. □

Proposition 20.7. *Every model is found.*

Proof. According to Proposition 20.4, EnumerateIrredundant terminates. The final state M is reached if the search space has been processed exhaustively. Hence, all models have been found. □

Note: Proposition 20.7 says that a sequence of all models of the formula is found. It therefore corresponds to completeness according to Definition 20.5 and to strong completeness according to Definition 20.6, if restarts are applied.

Proposition 20.8. *Every model is found exactly once.*

Proof. We recall Proposition 20.1 stating that only pairwise contradicting models are detected. In essence, this says that every model is found exactly once. □

Theorem 20.1 (Correctness). *If* $(P,\ N,\ 0,\ \varepsilon,\ \delta_0) \rightsquigarrow^*_{\mathsf{EnumIrred}} M$*, then*

(i) $M \equiv \pi(F, X)$

(ii) $C_i \wedge C_j \equiv 0$ *for* $C_i, C_j \in M$ *and* $C_i \neq C_j$

Proof. The cubes in M are exactly the $I^\star$ computed from the total models of P. These are models of $\pi(P, X)$ (Proposition 20.5). Since by Proposition 20.7 all models are found, $M \equiv \pi(P, X)$. But by Equation 20.2 $\mathsf{models}(\exists\, Y, S\,.\, P(X, Y, S)) = \mathsf{models}(\exists\, Y\,.\, F(X, Y))$ holds, i. e.$\mathsf{models}(\pi(P, X)) = \mathsf{models}(\pi(F, X))$. Thus,$M \equiv \pi(F, X)$, and (i) holds.

Due to Proposition 20.1, the found models are pairwise contradicting, and (ii) holds as well. Notice that one could also use Proposition 20.8, since as its consequence, only pairwise contradicting models are found. □

20.8.4 *Generalization to Partial Model Detection*

EnumerateIrredundant only finds total models of P. In SAT solving, this makes sense from an computational point of view, because checking whether a partial assignment satisfies a formula is more expensive than extending it to a total one. However, model enumeration is computationally more expensive than SAT solving, hence satisfiability checks, e. g., in the form of entailment checks [141], might pay off. Notice that it still might make sense to shrink the models found. In this section, we discuss the changes to be made to our presentation in order to support the detection of partial models.

First, the satisfiability condition need be replaced. This affects "all variables in $X \cup Y \cup S$ are assigned" on line 15 of EnumerateIrredundant (Figure 20.1) and "$(X \cup Y \cup S) - I = \emptyset$" in rules End1 and Back1 in our calculus (Figure 20.3). These conditions are replaced by "I is a model of P" and "$I(P) = 1$", respectively.

Second, some proofs need slightly be adapted. In the proof of Proposition 20.3 we use the assumption that I is total to justify that a conflict in $N|_I$ is obtained by propagating only variables in T. Now Invariant InvDualPN ensures that a conflict in $N|_I$ is obtained also if I is a partial assignment. However, it might be the case that input variables might need be propagated or decided, and we might obtain some I' containing literals which do not occur on I. Defining $I^\star = \pi(I', \mathsf{var}(I))$ solves the issue, and the rest of the proof remains unchanged. Notice that this change is reflected neither in EnumerateIrredundant nor in the calculus, since the computation of $I^\star$ is not specified. Adequate changes need be done in the wording and in the proof of Proposition 20.5. The rest of the proof remains unaffected.

20.9 CONFLICT-DRIVEN CLAUSE LEARNING FOR REDUNDANT ALL-SAT

Let F be a formula in CNF over a set of variables V, and suppose the current trail I satisfies F. Let the last decision literal on I be $\neg\ell$. It is flipped and the search continues. This corresponds to learning the negation of I and backtracking chronologically, i. e., to the previous decision level, after which $\neg I$ becomes unit and its unit literal ℓ is propagated. The clause $\neg I$ acts as a reason for ℓ. If at a later stage of the search, a conflict occurs, it is analyzed, and a clause blocking this falsifying assignment is learned.

Conflict-driven clause learning (CDCL) is based on the assumption that the reason of every literal which is not a decision literal is contained in the formula. If blocking clauses are added to the input formula, this is indeed the case, and CDCL for model enumeration does not differ from its SAT counterpart. However, this is not the case anymore if no blocking clauses are used. The question now is how ℓ should be treated if its reason $\neg I$ was not added to the formula.

We propose to consider ℓ as a propagated literal annotating it on the trail with its reason $\neg I$ but without adding $\neg I$ to F. This ensures the working of conflict analysis. Notice that $\neg I$ can not be used for unit propagation. Furthermore, any clause learned in conflict analysis involving $\neg I$ is logically entailed by $F \wedge \neg I$ but not necessarily by F, to which it is added. This is in contrast to CDCL for SAT, where the clauses learned after a conflict are entailed by F. The idea is best conveyed by a small example.

Example 20.8 (Conflict analysis for model enumeration)**.** *Consider the formula F over the set of variables $V = \{a, b, c, d, e\}$:*

$$F = \underbrace{(a \vee b \vee \neg c)}_{C_1} \wedge \underbrace{(\neg b \vee c)}_{C_2} \wedge \underbrace{(d \vee \neg c \vee e)}_{C_3} \wedge \underbrace{(d \vee \neg c \vee \neg e)}_{C_4}$$

Assume $X = \{a, b, c, d\}$ and $Y = \{e\}$. Suppose we decide a and b, propagate c with reason C_2 and decide d followed by deciding e. The resulting trail $I_1 = a^d\, b^d\, c^{C_2}\, d^d\, e^d$ is a model of F. This model is blocked by $B_1 = (\neg a \vee \neg b \vee \neg d)$ consisting of the negated relevant decision literals on I_1. Considering only the decisions ensures that B_1 contains exactly one literal per decision level and that after backtracking chronologically B_1 becomes unit. Recall that B_1 is not added to F. The last decision literal is flipped with reason B_1 and e is propagated with reason C_3. But now C_4 is falsified. The current trail is $I_2 = a^d\, b^d\, c^{C_2}\, \neg d^{B_1}\, e^{C_3}$, which can also be visualized by the following implication graph:

$$\begin{tikzcd}
a@1 \arrow[r, "B_1"] & \neg d \arrow[dr, "C_3"] \arrow[drr, bend left, "C_4"] & & \\
 & & e \arrow[r, "C_4"] & \kappa \\
b@2 \arrow[uur, "B_1"] \arrow[r, "C_2"'] & c \arrow[ur, "C_3"'] \arrow[urr, bend right, "C_4"'] & &
\end{tikzcd}$$

The nodes on the left hand side having no incoming edge represent decisions and are annotated by their decision level. The other nodes denote propagated literals whose reasons label their incoming edges. The node κ represents a conflict, and the conflicting clause is the one labeling its incoming edges.

The resolution steps for determining the clause representing the reason for the conflict can either be read off the trail I_2 in reverse assignment order or determined from the incident graph by following the arrows in reverse direction starting with κ. We first resolve the conflicting clause C_4 with the reason of e, C_3, obtaining the resolvent $(d \vee \neg c)$. Both d and $\neg c$ have the highest decision level 2, and we continue by resolving the resolvent with B_1 obtaining $(\neg c \vee \neg a \vee b)$, followed by resolution with C_2 resulting in $C_5 = (\neg a \vee \neg b)$, which has only one literal at decision level 2. The resolution process stops, and C_5 is added to F. Notice that learning a clause containing one single literal at conflict level 2 requires resolution with B_1.

To lay the focus on the main topic, namely conflict analysis in the absence of blocking clauses, in this section we consider neither model shrinking nor projection. However, the extension of the clause learning algorithm to support the two, is straightforward.

20.10 PROJECTED REDUNDANT MODEL ENUMERATION

Now we turn our attention to the case where enumerating models multiple times is permitted. This allows for refraining from adding blocking clauses to the formula under consideration, since they might significantly slow down the enumerator. This affects both our algorithm and our calculus for irredundant projected model enumeration. Omitting the use of blocking clauses has a minor impact on our algorithm and its formalization. For this reason, in this section we point out the differences between the two methods.

20.10.1 *Algorithm and Calculus*

The only difference compared to EnumerateIrredundant consists in the fact that no blocking clauses are added to P. However, they are remembered as annotations on the trail in order to enable conflict analysis after finding a model. Our algorithm EnumerateRedundant therefore is exactly the same as EnumerateIrredundant listed in Figure 20.1 without lines 22–23. The annotation of flipped literals happens in function Backtrack() in line 26.

Accordingly, our formalization consists of all rules of the calculus in Figure 20.3 but replacing rule Back1 by rule Back1red shown in Figure 20.6. Rule Back1red differs from rule Back1 only in the fact that both P and N remain unaltered.

$$\begin{array}{ll}
\mathsf{Back1red}: & (P,\ N,\ M,\ I,\ \delta) \rightsquigarrow_{\mathsf{Back1red}} (P,\ N,\ M \vee m,\ J\ell^{B},\ \delta[K \mapsto \infty][\ell \mapsto b]) \\
& \text{if}\ \ (X \cup Y \cup S) - I = \varnothing\ \ \text{and}\ \ \text{exists}\ I^{\star} \leqslant \pi(I, X \cup Y)\ \ \text{with} \\
& JK = I\ \ \text{such that}\ \ N \wedge I^{\star} \vdash_1 0\ \ \text{and}\ \ m \overset{\text{def}}{=} \pi(I^{\star}, X)\ \ \text{and} \\
& B \overset{\text{def}}{=} \neg\mathsf{decs}(m)\ \ \text{and}\ \ b+1 \overset{\text{def}}{=} \delta(B) = \delta(m)\ \ \text{and}\ \ \ell \in B\ \ \text{and} \\
& \ell|_K = 0\ \ \text{and}\ \ b = \delta(B \setminus \{\ell\}) = \delta(J) \\
\mathsf{Back0red}: & (P,\ N,\ M,\ I,\ \delta) \rightsquigarrow_{\mathsf{Back0red}} (P \wedge D^{r},\ N,\ M,\ J\ell^{D},\ \delta[K \mapsto \infty][\ell \mapsto j]) \\
& \text{if}\ \ \text{exists}\ C \in P\ \ \text{and}\ \ \text{exists}\ D\ \ \text{with}\ \ JK = I\ \ \text{and}\ \ C|_I = 0\ \ \text{and} \\
& \delta(C) = \delta(D) > 0\ \ \text{such that}\ \ \ell \in D\ \ \text{and}\ \ \neg\ell \in \mathsf{decs}(I)\ \ \text{and} \\
& \neg\ell|_K = 0\ \ \text{and}\ \ P \wedge \neg M \models D\ \ \text{and}\ \ j \overset{\text{def}}{=} \delta(D \setminus \{\ell\}) = \delta(J)
\end{array}$$

Figure 20.6: Rules for backtracking after detection of a model in redundant model enumeration. The calculus for redundant projected model enumeration differs from its irredundant counterpart only in the fact that no blocking clauses are used. Hence, all rules in Figure 20.3 are maintained except for rules Back1 and Back0, which are replaced by rules Back1red and Back0red, respectively.

20.10.2 *Example*

Example 20.9 (Projected redundant model enumeration)**.** *Consider again Example 20.1 elaborated in detail in Section 20.8.2 for* EnumerateIrredundant*. We have*

$$P = \underbrace{(a \vee c)}_{C_1} \wedge \underbrace{(a \vee \neg c)}_{C_2} \wedge \underbrace{(b \vee d)}_{C_3} \wedge \underbrace{(b \vee \neg d)}_{C_4}$$

and

$$\begin{aligned}
N = {} & \underbrace{(\neg t_1 \vee \neg a)}_{D_1} \wedge \underbrace{(\neg t_1 \vee \neg c)}_{D_2} \wedge \underbrace{(a \vee c \vee t_1)}_{D_3} \wedge \\
& \underbrace{(\neg t_2 \vee \neg a)}_{D_4} \wedge \underbrace{(\neg t_2 \vee c)}_{D_5} \wedge \underbrace{(a \vee \neg c \vee t_2)}_{D_6} \wedge \\
& \underbrace{(\neg t_3 \vee \neg b)}_{D_7} \wedge \underbrace{(\neg t_3 \vee \neg d)}_{D_8} \wedge \underbrace{(b \vee d \vee t_3)}_{D_9} \wedge \\
& \underbrace{(\neg t_4 \vee \neg b)}_{D_{10}} \wedge \underbrace{(\neg t_4 \vee d)}_{D_{11}} \wedge \underbrace{(b \vee \neg d \vee t_4)}_{D_{12}} \wedge \\
& \underbrace{(t_1 \vee t_2 \vee t_3 \vee t_4)}_{D_{13}}
\end{aligned}$$

Suppose $X = \{a, b\}$ and $Y = \{c, d\}$. The execution trail is depicted in Table 20.2.

Assume we decide a, b, c, and d (steps 1–4) obtaining the trail $I_1 = a^d\, b^d\, c^d\, d^d$, which is a model of P. Dual model shrinking occurs as in step 5 in the example elaborated in Section 20.8.2, except that the assumed literals b and c occur in a different order, and the same model a b is obtained. Notice that the clause $B_1 = (\neg a \vee \neg b)$ is not added to P.

After backtracking, we have $P|_{I_1} = (d) \wedge (\neg d)$, and after propagating d (step 6), we obtain a conflict. The current trail is $I_3 = a^d\, \neg b^{B_1}\, d^{C_3}$, and $C_4|_{I_3} = ()$. Resolution of the reasons on I_3 in reverse assignment order is executed, starting with the conflicting

Table 20.2: Execution trace for $F = (a \vee c) \wedge (a \vee \neg c) \wedge (b \vee d) \wedge (b \vee \neg d)$ defined over the set of relevant variables $X = \{a, b\}$ and the set of irrelevant variables $Y = \{c, d\}$ (see also Example 20.1).

STEP	RULE	I	$P\|_I$	M
0		ε	P	0
1	DecX	a^d	$(b \vee d) \wedge (b \vee \neg d)$	0
2	DecX	$a^d\, b^d$	1	0
3	DecYS	$a^d\, b^d\, c^d$	1	0
4	DecYS	$a^d\, b^d\, c^d\, d^d$	1	0
5	Back1red	$a^d\, \neg b^{(\neg a \vee \neg b)}$	$(d) \wedge (\neg d)$	$a \wedge b$
6	Unit	$a^d\, \neg b^{(\neg a \vee \neg b)}\, d^{C_3}$	0	$a \wedge b$
7	Back0	$b^{(b)}$	$(a \vee c) \wedge (a \vee \neg c)$	$a \wedge b$
8	DecX	$b^{(b)}\, a^d$	1	$a \wedge b$
9	DecYS	$b^{(b)}\, a^d\, c^d$	1	$a \wedge b$
10	DecYS	$b^{(b)}\, a^d\, c^d\, d^d$	1	$a \wedge b$
11	Back1red	$b^{(b)}\, \neg a^{(\neg b \vee \neg a)}$	$(c) \wedge (\neg c)$	$(a \wedge b) \vee (b \wedge a)$
12	Unit	$b^{(b)}\, \neg a^{(\neg b \vee \neg a)}\, c^{C_1}$	0	$(a \wedge b) \vee (b \wedge a)$
13	End0			$(a \wedge b) \vee (b \wedge a)$

clause C_4. We obtain $C_4 \otimes C_3 = (b) = C_5$, which contains exactly one literal at the maximum decision level, hence no further resolution steps are required. Since (b) is unit, the enumerator backtracks to decision level 0 and propagates b with reason C_5 (step 7). After deciding a, c, and d, we find the same model $b\,a\,c\,d$ as in step 4 (steps 8–10). Obviously, model shrinking provides us with the same model $b\,a$, which is added to M, and the last relevant decision is flipped (step 11). Now unit propagation leads to a conflict (step 12), and since there are no decisions on the trail, the procedure stops (step 13). Now the cubes in M, which represent the models of P, are not pairwise disjoint anymore. However, we still have $M \equiv \pi(P, X) \equiv \pi(F, X)$.

20.10.3 *Proofs*

Invariants InvDualPN and InvDecs listed in Figure 20.4 are applicable also for redundant model enumeration, since they involve none of P and N. Invariant InvImplIrred instead need be adapted since no blocking clauses are added to P, and therefore it ceases to hold. Assume a model I has been found and shrunken to m, and that the last relevant decision literal ℓ has been flipped. Since its reason $B = \mathsf{decs}(\neg m)$ is not added to P, from $P \wedge \mathsf{decs}(I)$ we can not infer I. Recall that instead m is added to M, hence $\neg M$ contains the reasons of all decision literals flipped after having found a model. A closer look reveals that this case is analog to the one in our previous work [139]. In this work, we avoided the use of blocking clauses by means of chronological backtracking. However, the basic idea is the same, and we replace Invariant InvImplIrred by Invariant InvImplRed listed in Figure 20.7. This is exactly Invariant (3) in our previous work on model

$$\text{InvDualPN: } \exists S\,[\,P(X,Y,S)\,] \equiv \neg\exists T\,[\,N(X,Y,T)\,]$$

$$\text{InvDecs: } \delta(\mathsf{decs}(I)) = \{1,\ldots,\delta(I)\}$$

$$\text{InvImplRed: } \forall n \in \mathbb{N}\,.\,P \wedge \neg M \wedge \mathsf{decs}_{\leqslant n}(I) \models I_{\leqslant n}$$

Figure 20.7: Invariants for projected model enumeration with repetition. Notice that Invariants InvDualPN and InvDecs are the same as for irredundant model enumeration, while due to the lack of blocking clauses, in invariant InvImplIrred the models recorded in M need be considered.

counting [139], hence in our proof we use a similar argument. The invariants for redundant model enumeration under projection are given in Figure 20.7.

20.10.3.1 *Invariants in Non-Terminal States*

Proposition 20.9 (Invariants in EnumerateRedundant)**.** *The Invariants InvDualPN, InvDecs, and InvImplRed hold in non-terminal states.*

Proof. The proof is carried out by induction over the number of rule applications. Assuming Invariants InvDualPN, InvDecs, and InvImplRed hold in a non-terminal state (P, N, M, I, δ), we show that they are met after the transition to another non-terminal state for all rules.

Now rules End1, End0, Back0, DecX, and DecYS are the same as for EnumerateIrredundant (Figure 20.3). In Section 20.8.3.1 we already proved that after the execution of these rules Invariants InvDualPN and InvDecs still hold.

As for invariant InvImplRed, from (i) and (ii) in Proposition 20.6 and observing that $m \leqslant I$, where I is a total model of P and $m \in M$ its projection onto the relevant variables, we can conclude that invariant InvImplRed holds as well. To see this, remember that in invariant InvImplIrred we consider $P = P_0 \wedge \bigwedge_i B_i$, where the B_i are the clauses added to P_0 blocking the models m_i. But $\neg B_i \leqslant m_i$, hence invariant InvImplRed holds after the application of the rules Unit, Back0red, DecX, and DecYS, and we are left to carry out the proof for rule Back1red.

<u>Back1red</u>

Invariant InvDualPN: Both P and N remain unaltered, therefore Invariant InvDualPN holds after the application of Back1red.

Invariant InvDecs: The proof is analogous to the one for rule Back1.

Invariant InvImplRed: We need to show $P \wedge \neg(M \vee m) \wedge \mathsf{decs}_{\leqslant n}(J\ell) \models (J\ell)_{\leqslant n}$ for all n. First notice that the decision levels of all the literals in J do not change while applying the rule. Only the decision level of the variable of ℓ is decremented from $b+1$ to b. It also stops being a decision. Since $\delta(J\ell) = b$, we can assume $n \leqslant b$. Observe that $P \wedge \neg(M \vee m) \wedge \mathsf{decs}_{\leqslant n}(J\ell) \equiv \neg m \wedge (P \wedge \neg M \wedge \mathsf{decs}_{\leqslant n}(I))$, since ℓ is not a decision in $J\ell$ and $I_{\leqslant b} = J$ and $I_{\leqslant n} = J_{\leqslant n}$ by definition. Now the induction hypothesis is applied and we get $P \wedge \neg(M \vee m) \wedge \mathsf{decs}_{\leqslant n}(J\ell) \models I_{\leqslant n}$. Again using $I_{\leqslant n} = J_{\leqslant n}$ this almost closes the proof except that we are left to prove $P \wedge \neg(M \vee m) \wedge \mathsf{decs}_{\leqslant e}(J\ell) \models \ell$ as ℓ has decision level b in $J\ell$ after applying the rule and thus ℓ disappears in the proof obligation for $n < b$. To see this notice that $P \wedge \neg B \models I_{\leqslant b+1}$ using again the induction hypothesis for $n = b+1$ and recalling that $\neg B = \mathsf{decs}_{\leqslant b+1}(I)$. This gives $P \wedge \neg\mathsf{decs}_{\leqslant b}(J) \wedge \neg\ell \models I_{\leqslant b+1}$ and thus $P \wedge \neg\mathsf{decs}_{\leqslant b}(J) \wedge \neg I_{\leqslant b+1} \models \ell$ by conditional contraposition.

□

20.10.3.2 *Progress and Termination*

The proofs that our method for redundant projected model enumeration always makes progress and eventually terminates are the same as in Section 20.8.3.2 and Section 20.8.3.3.

20.10.3.3 *Equivalence*

Some properties proved for the case of irredundant model enumeration cease to hold if we allow enumerating redundant models. Specifically, Proposition 20.5, and Proposition 20.7 hold, while Proposition 20.8 does not. Both (i) and (ii) of Proposition 20.6 hold, while (iii) does not. In Theorem 20.1, (i) holds but (ii) does not. Their proofs remain the same as for irredundant model enumeration in Section 20.8.3.1.

20.10.4 *Generalization*

The same observations apply as for irredundant model enumeration presented in Section 20.8.4.

20.11 CONCLUSION

Model enumeration and projection, with and without repetition, is a key element to several tasks. We have presented two methods for propositional model enumeration under projection. EnumerateIrredundant uses blocking clauses to avoid enumerating models multiple times, while EnumerateRedundant is exempt from blocking clauses and admits repetitions. Our CDCL-based model enumerators detect total models and uses dual reasoning to shrink them.

To ensure correctness of the shrinking mechanism, we developed a dual encoding of the blocking clauses. We provided a formalization and proof of correctness of our blocking-based model enumeration approach and discussed a generalization to the case where partial models are found. These partial models might not be minimal, hence shrinking them still might make sense. Also, there is no guarantee that the shrunken models are minimal as they depend on the order of the variable assignments.

We presented a conflict-driven clause learning mechanism for redundant model enumeration, since standard CDCL might fail in the absence of blocking clauses. Basically, those clauses are remembered on the trail without being added to the input formula. This prevents a blowup of the formula but also does not further make use of these potentially short clauses, which in general propagate more eagerly than long clauses.

We discussed the modifications of our blocking-based algorithm and calculus to support redundant model enumeration and provided a correctness proof. Intuitively, shorter partial models representing non-disjoint sets of total models might be found.

Our method does not guarantee that the shrunken model $I^\star$ is minimal w.r.t. the decision level b in line EnumerateIrredundant. However, finding short DSOPs is important in circuit design [136], and appropriate algorithms have been introduced by, e. g., Minato [135]. While DSOP minimization has been proven to be NP-complete [17], finding a smaller decision level *b* would already be advantageous, since besides restricting the search space to be explored it generates shorter models. To this end, we plan to adapt our dual shrinking algorithm to exploit the Tseitin encoding as proposed by Iser et al. [101].

In the presence of multiple conflicting clauses, a related interesting question might also be which one to choose as a starting point for conflict analysis with the aim to backtrack as far as possible. This is not obvious unless all conflicts are analyzed.

Determining shortest possible models makes our approach suitable for circuit design. We are convinced that this work provides incentives not only for the hardware-near community but also for the enumeration community.

21

DISCUSSION OF PAPER 6

The main contributions are highlighted in Section 21.1. The adaptation of strong completeness in the absence of reasons for flipped decisions is clarified by an example in Section 21.2. In this setting, a clause learnt by conflict analysis is entailed not only by the input formula but also by the clauses representing the reasons of flipped decision literals. We demonstrate this by an example in Section 21.3.

21.1 MAIN CONTRIBUTIONS

Based on their counterparts in the context of SAT solving [197], soundness, completeness, and strong completeness are defined for propositional model enumeration. We claim that our definitions are applicable also for methods detecting partial models. While this can easily be seen for soundness and completeness, it is less obvious for strong completeness, and we come back to strong completeness in the context of partial model enumeration in Section 21.2.

A method for shortening total models is introduced. The total models are obtained by standard CDCL with conflict-driven backjumping, and for shrinking them we fall back on dual reasoning [137]. Basically, a second SAT solver is invoked incrementally on the negation of the formula and the total satisfying assignment. A conflict is obtained by unit propagation, and conflict analysis is executed to determine the assignments involved in the conflict. These assignments constitute a partial model of the formula. Limiting the usage of dual reasoning to model shrinking avoids the additional work introduced by processing the input formula and its negation simultaneously observed in our earlier work (Chapter 11).

Non-chronological backtracking after finding a model requires the use of blocking clauses if repetitions must be avoided. Our dual model shrinking method works on the negation of the current formula, which includes the blocking clauses added so far. To block a model in the negated formula, it must be added with a disjunction to it (see also Section 20.6). We present a dual blocking clause encoding to maintain the negated formula in CNF. Dual blocking clauses enable us to use blocking clauses in a dual setting solving an issue identified in our framework for dual model counting (Section 11.4).

Blocking clauses provide the reasons of flipped decision literals (see also Chapter 7). In redundant model enumeration, where models may be reported multiple times, they are not required. Consequently, the conflict analysis procedure need be adapted. We propose to annotate decision literals flipped after finding a model with the corresponding blocking clause but without adding it to the formula. This avoids issues during later conflict analysis due to missing reasons. However, since these clauses are not added to the formula, they are not used for unit propagation.

We present two algorithms and their formalization for model enumeration under projection, both making use of dual model shrinking and non-chronological backtracking after a model. The first framework addresses irredundant model enumeration. It returns a DSOP formula and hence a d-DNNF formula and can therefore readily be adapted to compute the model count of the input formula. Repetitions are avoided by means of blocking clauses. The second framework addresses redundant model enumeration. It is exempt from the use of blocking clauses and therefore adopts our variant of conflict analysis mentioned above. Generalizations to detecting and shrinking partial models are described.

The invariants met by our approaches were already introduced in our earlier work presented in Section 14.5 and Section 16.5. We only want to notice that Invariant InvDualPN corresponds to the duality property introduced in Section 11.3. Finally, we provide proofs of correctness of our frameworks.

21.2 STRONG COMPLETENESS IN PARTIAL MODEL ENUMERATION

Strong completeness says that the model enumerator is guaranteed to find "any arbitrary sequence containing all models of a satisfiable formula" (Definition 20.6). Obviously, the adopted model enumeration algorithm has to be considered. Assume a formula F has two partial models m_1 and m_2, besides others, which represent two non-disjoint sets of total models of F. A model enumerator using blocking clauses, which is sometimes also said to be *blocking*, finds either m_1 or m_2 during one single execution but not both. Similarly, it is not guaranteed that a non-blocking solver enumerates all partial models, although this might happen. The point is that every total model of F is represented by the partial models returned by the model enumerator, and thinking of partial models as sets of total models, the enumerator—be it blocking or non-blocking—finds any sequence of *total* models of F. Example 21.1 clarifies this idea.

Example 21.1 (Strong completeness in partial model enumeration)**.** *Consider the formula $F = (a \lor c) \land (b \lor d)$ defined over the set of variables $V = \{a, b, c, d\}$. Its total models are $a\,b\,c\,d$, $a\,b\,c\,\neg d$, $a\,b\,\neg c\,d$, $a\,b\,\neg c\,\neg d$, $a\,\neg b\,c\,d$, $a\,\neg b\,\neg c\,d$, $\neg a\,b\,c\,d$, $\neg a\,b\,c\,\neg d$, and $\neg a\,\neg b\,c\,d$. Its partial models are $a\,b$, $a\,d$, $b\,c$, $c\,d$, $a\,b\,c$, $a\,b\,\neg c$, $a\,b\,d$, $a\,b\,\neg d$, $a\,\neg b\,d$, $a\,c\,d$, $a\,\neg c\,d$, $\neg a\,b\,c$, $b\,c\,d$, $b\,c\,\neg d$, $\neg a\,c\,d$, and $\neg b\,c\,d$. A non-blocking model enumerator might enumerate $a\,b$ and $c\,d$. If a blocking model enumerator finds $a\,b$, in the next model one of a and b occurs negatively, hence it can not happen that it finds all partial models of F in one run. However, in another run it might find a different set of partial models.*

21.3 LEARNT CLAUSES IN REDUNDANT MODEL ENUMERATION

Clauses learnt by means of conflict analysis are obtained by a sequence of resolution steps. As such, they are entailed by the formula, i. e., all assignments satisfying the formula also satisfy the learnt clauses, and this obviously includes all blocking clauses added to the formula up to the point where the conflict occurred.

Conflict analysis assumes that the reason of every propagated literal occurs in the formula. In our redundant model enumeration approach, we refrain from using blocking clauses but remember them in order to avoid issues with later conflict analysis. Consequently, a clause obtained by conflict analysis is entailed by the conjunction of the formula and all clauses ever used for annotating propagated literals on the trail, as is shown by means of an example.

Example 21.2. *Consider again the situation in Example 20.8, where*

$$F(X,Y) = \underbrace{(a \vee b \vee \neg c)}_{C_1} \wedge \underbrace{(\neg b \vee c)}_{C_2} \wedge \underbrace{(d \vee \neg c \vee e)}_{C_3} \wedge \underbrace{(d \vee \neg c \vee \neg e)}_{C_4}$$

is defined over the set of relevant variables $X = \{a, b, c, d\}$ and the set of irrelevant variables $Y = \{e\}$. The total models of F projected onto X are $a\,b\,c\,d$, $a\,\neg b\,c\,d$, $a\,\neg b\,\neg c\,d$, $a\,\neg b\,\neg c\,\neg d$, $\neg a\,b\,c\,d$, $\neg a\,\neg b\,\neg c\,d$, and $\neg a\,\neg b\,\neg c\,\neg d$.

The trail $I_1 = a^d\,b^d\,c^{C_2}\,d^d\,e^d$ satisfies F. The most recent relevant decision literal d^d is flipped and the corresponding clause $B_1 = (\neg a \vee \neg b \vee \neg d)$ remembered as its reason. After propagating e with reason C_3, a conflict in C_4 is obtained. For learning the conflict clause, first C_4 is resolved with C_3 followed by resolution with B_1 and C_2, and the clause $C_5 = (\neg a \vee \neg b)$ is learnt (see Example 20.8 for a visualization of the implication graph and resolution steps). The computation of C_5 also involved resolution with B_1, and therefore $F \wedge B_1 \models C_5$, as can be seen by looking at the total models of $F \wedge B_1$ projected onto X: these are $a\,\neg b\,c\,d$, $a\,\neg b\,\neg c\,d$, $a\,\neg b\,\neg c\,\neg d$, $\neg a\,b\,c\,d$, $\neg a\,\neg b\,\neg c\,d$, and $\neg a\,\neg b\,\neg c\,\neg d$, and all of them satisfy C_5. The total models of $F \wedge B_1$ are exactly the total models of F projected onto X without $a\,b\,c\,d$, which falsifies C_5.

We therefore aim at a version of CDCL taking into account blocking clauses for conflict analysis but without adding them to the formula. These blocking clauses are not used for unit propagation, and they are no candidates for conflicts, either.

Part VI

CONCLUSION

22

DISCUSSION

The aim of this thesis was to develop alternative methods to the component-based approach representing the state of the art in propositional model counting or #SAT. In the course of our work, we extended our investigations to All-SAT, the task of enumerating the models of a propositional formula. Both tasks are computationally expensive due to the size of the search space which—in contrast to SAT—need be processed exhaustively. Pruning the search space was therefore a major concern, and we focus on identifying short partial models. This contrasts component-based model counting, since search space pruning in this paradigm is achieved by processing individual components which are defined over a subset of the variables occurring in the original formula. On the other hand, the total extensions of partial models need not be checked explicitly, because they also satisfy the input formula. Short partial models therefore bear the potential to considerably prune the search space and thus reduce the amount of work.

Our first approach to model counting is dual and considers the formula under consideration together with its negation. The idea to exploit duality in Boolean logic is not new. In fact, prior to the work on dual reasoning in quantified Boolean formula (QBF) solving, it has been adopted in the context of Boolean circuits [4, 91] and satisfiability modulo theories (SMT) for computing interpolants for quantifier-free equalities and uninterpreted functions [2]. Our approach differs from the dual propagation method by Goultiaeva and Bacchus [91] in that it works on two formulae instead of propagating primal and dual values on the same circuit. Amarù, Gaillardon, Mishchenko, Ciesielski, and De Micheli [4] showed that a circuit representing a contradiction can be converted into a tautology and vice versa without the use of exact logical negation. In contrast, we use the exact negation of the input formula as its dual and conclude that the residual of the input formula F under a trail I is a tautology, if a conflict in the residual of its negation $\neg F$ under I occurs.

The motivation behind the work of Goultiaeva and Bacchus [91] was to enhance the power of cube learning similarly to clause learning in CDCL. While they succeeded in this goal, it turned out that our dual approach introduced in Chapter 11 is not effective on CNF instances, since the additional work introduced by processing two formulae could not be compensated by the detection of shorter partial models. Similarly to their work, conflicts in N might allow for saving an exponential amount of work. However, we can not take advantage of CDCL with non-chronological backtracking after a conflict in N, since models might be missed, and therefore our approach is not totally symmetric unlike the one adopted in the QBF solver IQTest [204], which works on formulae in Combined Conjunctive-Disjunctive Normal Form (CCDNF). These formulae consist of a CNF and a DNF part, which are defined over different sets of variables. This representation allows for a symmetric reasoning in clauses and cubes but can not readily be processed by modern SAT-based model counters, since they mostly work on CNF formulae.

The main gain of dual reasoning in the context of model counting is that it allows for detecting potentially short partial models of the input formula F by exploiting the capability of CDCL-based SAT solvers to detect conflicts. An assignment falsifying $\neg F$ is a (partial) model of F and allows for pruning the search space without the need for executing satisfiability checks or implementing clause watching mechanisms. Unit propagation in the negated formula is executed in place of taking and flipping a decision saving further work. We crafted a formula which does not easily decompose and showed that our dual model counter Dualiza outperforms the component-based state-of-the-art #SAT solvers on this instance. The reason is a fundamental one, and we conjecture that this observation holds for all formulae which can not easily be partitioned into subformulae over disjoint sets of variables.

Further work is saved by combining CDCL with chronological backtracking, or chronological CDCL. Our investigation and formalization of chronological CDCL provided us with a deeper understanding of CDCL and chronological backtracking. Chronological CDCL was implemented in the SAT solver CaDiCaL, and experiments showed that even if we always backtrack chronologically, the solver performance does not degrade much. Since then, chronological CDCL became a permanent part of CaDiCaL,[1] which won the first place in the SAT track of the SAT Race 2019[2] and second overall place, and kissat,[3] which won the main track of the SAT Competition 2020.[4]

Chronological CDCL bears the potential to be useful in model counting, and we provide a formalization and proofs of a model counting approach based on chronological CDCL. In this work, we focused on theory and did not provide an implementation. Both model counting based on chronological CDCL and dual model counting compute residuals in order to determine whether a partial assignment I satisfies the input formula F, i. e., execute a syntactic satisfiability check. However, in our dual approach a model of F might be detected which does not satisfy the CNF representation of F but its projection onto the set of relevant variables. This is particularly interesting in the presence of auxiliary variables. The models detected by model counting based on chronological CDCL instead always satisfy the input formula, which is assumed to be in CNF, and since projection is not supported, it may not contain auxiliary variables. Finally, we can not use chronological CDCL to deal with a conflict in $\neg F$, as models of F might be lost.

Testing whether a partial trail I already satisfies a formula F before taking a decision results in finding the shortest models possible in a non-dual setting. Consider again the precise entailment condition $\forall X \exists Y \, [\, F|_I \,] = 1$, where $F(X, Y)$ is a propositional formula defined over the set of relevant variables X and the set of irrelevant variables Y and I denotes a trail over variables in $X \cup Y$. It enables the detection of partial assignments represented by a trail I, whose total extensions are models of F, while $F|_I \neq 1$. Concretely, the semantic check $F|_I \equiv 1$ is carried out in contrast to the syntactic check $F|_I = 1$, which is used in our previous work. Therefore, shorter models are found than by model counting based on chronological CDCL. These models are also shorter than the ones detected by dual model counting, since they also relay on a syntactic check. Finally, blocking clauses used

1 https://github.com/arminbiere/cadical
2 http://sat-race-2019.ciirc.cvut.cz/
3 https://github.com/arminbiere/kissat
4 https://satcompetition.github.io/2020/

in combination with logical entailment tests will be shorter and therefore propagate more easily and prune larger portions of the search space.

Our last approach consists in shrinking total models using dual reasoning. It is not clear whether this leads to find shorter models than by our dual framework. However, since the satisfiability check is syntactic, testing for logical entailment before taking a decision might give us shorter models. The overhead introduced by shrinking total models will be reduced compared to our dual methods, since only one formula is processed at once. However, the third flavor of our entailment check involves a check for unsatisfiability, while dual model shrinking is known to lead to a conflict, and hence the latter we assume the latter to be less expensive unless a great number of blocking clauses is added, since they slow down unit propagation. Our dual blocking clause encoding enables their use in a dual setting, and its computation after finding a model is not too expensive under a theoretical aspect. In the case of redundant model enumeration their computation is obsolete, and the negative impact on solver performance due to a blowup of the input formula and its negation is avoided. Allowing repetitions might result in finding shorter partial models. This contrasts our observation that the longer the enumerator or counter runs the longer the enumerated models, e. g., in our dual counting framework. Finally, the conflict analysis variant proposed in the context of model enumeration exempt from blocking clauses prevents the solver to use them for unit propagation. It is not clear, however, whether this drawback can be made up by the gain obtained by refraining from the addition of blocking clauses.

Our tool DualCountPro allows for systematically trying our different variants of our framework and for identifying features which are not needed or which need always be assigned 1. As an example, we would never allow to split on dual variables, hence the feature `D_T` could just be omitted or assigned permanently 0.

23

SUMMARY

We have presented various approaches for counting or enumerating the models of a propositional formula with or without repetitions, some of which support projection. In the following, we are going to focus on our contributions in the fields of SAT solving and #SAT and All-SAT. As far as model detection is concerned, we consider model counting and model enumeration related, and the techniques we have presented are adoptable in either of the two. Our contributions to #SAT and All-SAT are therefore summarized in the same paragraph.

CONTRIBUTIONS TO PROPOSITIONAL SATISFIABILITY. We have presented a formalization of chronological CDCL, which combines CDCL and chronological backtracking. Their combination is challenging, since various invariants considered crucial to CDCL cease to hold. Our formalization and proof of correctness is a step towards a deeper understanding of the impact of chronological backtracking on the working of a CDCL-based SAT solver. We provide an implementation of chronological CDCL in CaDiCaL. Our experimental results motivated Armin Biere to make chronological CDCL a permanent part of CaDiCaL and Kissat.

CONTRIBUTIONS TO MODEL COUNTING AND ENUMERATION. We have developed various methods for detecting short partial models. Short models facilitate the pruning of a larger portion of the search space, and the corresponding blocking clauses propagate more easily. Our dual model framework considers the input formula and its negation and records any assignment either satisfying the formula or falsifying its negation. Due to the bias of SAT solvers towards finding conflict, the latter might happen earlier. Our tool Dualiza outperforms the component-based #SAT solver sharpSAT by orders of magnitude on a crafted formula which does not easily decompose. However, we conjecture that the reason is fundamental and that this behavior can be observed for all instances which do not decompose easily. We further propose to check whether the current partial assignment already satisfies the input formula before taking a decision. This method might result in finding ever shorter models, as the precise entailment check is a semantic one, in contrast to the syntactic check in our dual framework. To capture cases where a cheaper test is sufficient, we introduce four entailment tests of different strengths and computational costs. Alternatively, total models can be shrunken by dual reasoning. This method is computationally less expensive than our dual model counting approach or the execution of entailment tests while taking advantage of the strengths of dual reasoning.

24 FUTURE WORK

In this thesis, different approaches to detecting partial models have been developed. It would be interesting to investigate combinations of those. As an example, dual blocking clauses should be added to our dual model counting framework. This would allow for using dual reasoning in combination with blocking clauses. Along this line, it might be interesting to determine conditions for using flipping and discounting with blocking clauses. The combination of logical entailment checks and blocking clauses is another combination worth exploring, as well as dual reasoning with chronological CDCL. The latter would reduce the negative impact on solver performance by the use of blocking clauses.

A fundamental question is when to use which entailment test in partial model enumeration. Choosing the right test as early as possible would save redundant work introduced by running superfluous entailment tests. Similarly, by examining the input formula, one might be able to detect when its models are more efficiently counted by using only its negation, as in our example where the formula consists of one single clause. To our best knowledge, a truly symmetric treatment of satisfying and falsifying assignment was not achieved yet. Dual reasoning might be a first step towards a symmetric treatment of models and counter-models.

Understanding the reasons for the different performance of the two versions of CaDiCaL reported in Table 15.1 could provide us with a deeper understanding of the working of CaDiCaL and maybe of SAT solvers in general, or with the impact of formula features on SAT solvers.

Finally, benchmarks for model counting or enumeration stemming from practical applications are needed. We are particularly interested in benchmarks which do not decompose easily. The benchmarks available from the SAT competitions do decompose, and as we have seen, Dualiza is not effective on these without projection. Extending our frameworks and Dualiza in particular to support component-based reasoning is an interesting topic for future work. It is an open question, however, whether all relevant instances do decompose or whether those which do not simply did not make it into the SAT competition. An answer to this question also influences the future research direction in propositional model counting and enumeration.

Our tool DualCountPro is not yet finished. Some features are still missing, such as clause learning. Furthermore, it will be a valuable tool to check future additions to our framework, such as support for component-based reasoning.

BIBLIOGRAPHY

[1] Dimitris Achlioptas and Panos Theodoropoulos. "Probabilistic Model Counting with Short XORs." In: *Theory and Applications of Satisfiability Testing – SAT 2017 – 20th International Conference, Melbourne, VIC, Australia, August 28–September 1, 2017, Proceedings*. Ed. by Serge Gaspers and Toby Walsh. Vol. 10491. Lecture Notes in Computer Science. Springer, 2017, pp. 3–19. DOI: 10.1007/978-3-319-66263-3_1.

[2] Leonardo Alt, Antti Eero Johannes Hyvärinen, Sepideh Asadi, and Natasha Sharygina. "Duality-based interpolation for quantifier-free equalities and uninterpreted functions." In: *2017 Formal Methods in Computer Aided Design, FMCAD 2017, Vienna, Austria, October 2–6, 2017*. Ed. by Daryl Stewart and Georg Weissenbacher. IEEE, 2017, pp. 39–46. DOI: 10.23919/FMCAD.2017.8102239.

[3] Antoine Amarilli, Florent Capelli, Mikaël Monet, and Pierre Senellart. "Connecting Knowledge Compilation Classes and Width Parameters." In: *Theory Comput. Syst.* 64.5 (2020), pp. 861–914. DOI: 10.1007/s00224-019-09930-2.

[4] Luca Gaetano Amarù, Pierre-Emmanuel Gaillardon, Alan Mishchenko, Maciej J. Ciesielski, and Giovanni De Micheli. "Exploiting Circuit Duality to Speed up SAT." In: *2015 IEEE Computer Society Annual Symposium on VLSI, ISVLSI 2015, Montpellier, France, July 8–10, 2015*. IEEE Computer Society, 2015, pp. 101–106. DOI: 10.1109/ISVLSI.2015.18.

[5] Carlos Ansótegui, Maria Luisa Bonet, Jesús Giráldez-Cru, and Jordi Levy. "Structure features for SAT instances classification." In: *Journal of Applied Logic* 23 (2017), pp. 27–39. DOI: 10.1016/j.jal.2016.11.004.

[6] Carlos Ansótegui, Maria Luisa Bonet, Jesús Giráldez-Cru, Jordi Levy, and Laurent Simon. "Community Structure in Industrial SAT Instances." In: *Journal of Artificial Intelligence Research (JAIR)* 66 (2019), pp. 443–472. DOI: 10.1613/jair.1.11741.

[7] Carlos Ansótegui, Jesús Giráldez-Cru, and Jordi Levy. "The Community Structure of SAT Formulas." In: *Theory and Applications of Satisfiability Testing – SAT 2012 – 15th International Conference, Trento, Italy, June 17–20, 2012. Proceedings*. Ed. by Alessandro Cimatti and Roberto Sebastiani. Vol. 7317. Lecture Notes in Computer Science. Springer, 2012, pp. 410–423. DOI: 10.1007/978-3-642-31612-8_31.

[8] Cyrille Artho, Armin Biere, and Martina Seidl. "Model-Based Testing for Verification Back-Ends." In: *Tests and Proofs – 7th International Conference, TAP@STAF 2013, Budapest, Hungary, June 16–20, 2013. Proceedings*. Ed. by Margus Veanes and Luca Viganò. Vol. 7942. Lecture Notes in Computer Science. Springer, 2013, pp. 39–55. DOI: 10.1007/978-3-642-38916-0_3.

[9] Gilles Audemard and Laurent Simon. “Refining Restarts Strategies for SAT and UNSAT.” In: *Principles and Practice of Constraint Programming – 18th International Conference, CP 2012, Québec City, QC, Canada, October 8–12, 2012. Proceedings*. Ed. by Michela Milano. Vol. 7514. Lecture Notes in Computer Science. Springer, 2012, pp. 118–126. DOI: 10.1007/978-3-642-33558-7_11.

[10] Rehan Abdul Aziz, Geoffrey Chu, Christian J. Muise, and Peter J. Stuckey. “#∃SAT: Projected Model Counting.” In: *Theory and Applications of Satisfiability Testing – SAT 2015 – 18th International Conference, Austin, TX, USA, September 24–27, 2015, Proceedings*. Ed. by Marijn Heule and Sean A. Weaver. Vol. 9340. Lecture Notes in Computer Science. Springer, 2015, pp. 121–137. DOI: 10.1007/978-3-319-24318-4_10.

[11] Fahiem Bacchus, Shannon Dalmao, and Toniann Pitassi. “Algorithms and Complexity Results for #SAT and Bayesian Inference.” In: *44th Symposium on Foundations of Computer Science (FOCS 2003), 11–14 October 2003, Cambridge, MA, USA, Proceedings*. IEEE Computer Society, 2003, pp. 340–351. DOI: 10.1109/SFCS.2003.1238208.

[12] Fahiem Bacchus, Shannon Dalmao, and Toniann Pitassi. “DPLL with Caching: A new algorithm for #SAT and Bayesian Inference.” In: *Electronic Colloquium on Computational Complexity (ECCC)* 10.003 (2003). URL: http://eccc.hpi-web.de/eccc-reports/2003/TR03-003/index.html.

[13] Fahiem Bacchus, Shannon Dalmao, and Toniann Pitassi. “Solving #SAT and Bayesian Inference with Backtracking Search.” In: *Journal of Artificial Intelligence Research (JAIR)* 34 (2009), pp. 391–442. DOI: 10.1613/jair.2648.

[14] Fahiem Bacchus and Jonathan Winter. “Effective Preprocessing with Hyper-Resolution and Equality Reduction.” In: *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5–8, 2003 Selected Revised Papers*. Ed. by Enrico Giunchiglia and Armando Tacchella. Vol. 2919. Lecture Notes in Computer Science. Springer, 2003, pp. 341–355. DOI: 10.1007/978-3-540-24605-3_26.

[15] Roberto J. Bayardo Jr. and Joseph Daniel Pehoushek. “Counting Models Using Connected Components.” In: *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence, July 30–August 3, 2000, Austin, Texas, USA*. Ed. by Henry A. Kautz and Bruce W. Porter. AAAI Press / The MIT Press, 2000, pp. 157–162. URL: http://www.aaai.org/Library/AAAI/2000/aaai00-024.php.

[16] Paul Beame and Vincent Liew. “New Limits for Knowledge Compilation and Applications to Exact Model Counting.” In: *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence, UAI 2015, July 12–16, 2015, Amsterdam, The Netherlands*. Ed. by Marina Meila and Tom Heskes. AUAI Press, 2015, pp. 131–140. URL: http://auai.org/uai2015/proceedings/papers/111.pdf.

[17] Anna Bernasconi, Valentina Ciriani, Fabrizio Luccio, and Linda Pagli. “Compact DSOP and Partial DSOP Forms.” In: *Theory of Computing Systems* 53.4 (2013), pp. 583–608. DOI: 10.1007/s00224-013-9447-2.

[18] Armin Biere. *The AIGER And-Inverter Graph (AIG) Format Version 20071012*. Tech. rep. 07/01. Altenbergerstr. 69, 4040 Linz, Austria: Institute for Formal Models and Verification, Johannes Kepler University, 2007.

[19] Armin Biere. “CaDiCaL, Lingeling, Plingeling, Treengeling and YalSAT Entering the SAT Competition 2018.” In: *Proceedings of SAT Competition 2018: Solver and Benchmark Descriptions*. Ed. by Marijn Heule, Matti Järvisalo, and Martin Suda. Vol. B-2018-1. Department of Computer Science Series of Publications B. Department of Computer Science, University of Helsinki, 2018, pp. 13–14. URL: https://helda.helsinki.fi/handle/10138/237063.

[20] Armin Biere. “CaDiCaL at the SAT Race 2019.” In: *Proceedings of SAT Race 2019 : Solver and Benchmark Descriptions*. Ed. by Marijn J.H. Heule Heule, Matti Järvisalo, and Martin Suda. Vol. B-2019-1. Department of Computer Science Report Series B. Department of Computer Science, University of Helsinki, 2019, pp. 8–9. URL: https://helda.helsinki.fi/handle/10138/306988.

[21] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. “Symbolic Model Checking without BDDs.” In: *Tools and Algorithms for Construction and Analysis of Systems, 5th International Conference, TACAS '99, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS'99, Amsterdam, The Netherlands, March 22–28, 1999, Proceedings*. Ed. by Rance Cleaveland. Vol. 1579. Lecture Notes in Computer Science. Springer, 1999, pp. 193–207. DOI: 10.1007/3-540-49059-0_14.

[22] Armin Biere, Katalin Fazekas, Mathias Fleury, and Maximilian Heisinger. “CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling Entering the SAT Competition 2020.” In: *Proceedings of SAT Competition 2020 : Solver and Benchmark Descriptions*. Ed. by Tomáš Balyo, Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda. Vol. B-2020-1. Department of Computer Science Report Series B. Department of Computer Science, University of Helsinki, 2019, pp. 50–52. URL: https://helda.helsinki.fi/handle/10138/318450.

[23] Armin Biere, Mathias Fleury, and Maximilian Heisinger. “CaDiCaL, Kissat, Paracooba Entering the SAT Competition 2021.” In: *Proceedings of SAT Competition 2021 : Solver and Benchmark Descriptions*. Ed. by Tomáš Balyo, Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda. Vol. B-2021-1. Department of Computer Science Report Series B. Department of Computer Science, University of Helsinki, 2019, pp. 10–13. URL: https://helda.helsinki.fi/handle/10138/333647.

[24] Armin Biere, Steffen Hölldobler, and Sibylle Möhle. “An Abstract Dual Propositional Model Counter.” In: *YSIP2 – Proceedings of the Second Young Scientist's International Workshop on Trends in Information Processing, Dombai, Russian Federation, May 16–20, 2017*. Ed. by Steffen Hölldobler, Andrey Malikov, and Christoph Wernhard. Vol. 1837. CEUR Workshop Proceedings. CEUR-WS.org, 2017, pp. 17–26. URL: http://ceur-ws.org/Vol-1837/paper5.pdf.

[25] Armin Biere, Matti Järvisalo, and Benjamin Kiesl. “Preprocessing in SAT Solving.” In: *Handbook of Satisfiability – Second Edition*. Ed. by Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh. Vol. 336. Frontiers in Artificial Intelligence and Applications. IOS Press, 2021, pp. 391–435. DOI: 10.3233/FAIA200992. URL: https://doi.org/10.3233/FAIA200992.

[26] Fabrizio Biondi, Michael A. Enescu, Annelie Heuser, Axel Legay, Kuldeep S. Meel, and Jean Quilbeuf. “Scalable Approximation of Quantitative Information Flow in Programs.” In: *Verification, Model Checking, and Abstract*

Interpretation – 19th International Conference, VMCAI 2018, Los Angeles, CA, USA, January 7–9, 2018, Proceedings. Ed. by Isil Dillig and Jens Palsberg. Vol. 10747. Lecture Notes in Computer Science. Springer, 2018, pp. 71–93. DOI: 10.1007/978-3-319-73721-8_4.

[27] Elazar Birnbaum and Eliezer L. Lozinskii. "The Good Old Davis-Putnam Procedure Helps Counting Models." In: *Journal of Artificial Intelligence Research (JAIR)* 10 (1999), pp. 457–477. DOI: 10.1613/jair.601.

[28] Jasmin Christian Blanchette, Mathias Fleury, and Christoph Weidenbach. "A Verified SAT Solver Framework with Learn, Forget, Restart, and Incrementality." In: *Automated Reasoning – 8th International Joint Conference, IJCAR 2016, Coimbra, Portugal, June 27–July 2, 2016, Proceedings*. Ed. by Nicola Olivetti and Ashish Tiwari. Vol. 9706. Lecture Notes in Computer Science. Springer, 2016, pp. 25–44. DOI: 10.1007/978-3-319-40229-1_4.

[29] Jasmin Christian Blanchette, Mathias Fleury, Peter Lammich, and Christoph Weidenbach. "A Verified SAT Solver Framework with Learn, Forget, Restart, and Incrementality." In: *J. Autom. Reason.* 61.1–4 (2018), pp. 333–365. DOI: 10.1007/s10817-018-9455-7.

[30] Bernhard Bliem and Matti Järvisalo. "Centrality Heuristics for Exact Model Counting." In: *31st IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2019, Portland, OR, USA, November 4–6, 2019*. IEEE, 2019, pp. 59–63. DOI: 10.1109/ICTAI.2019.00017.

[31] Simone Bova, Florent Capelli, Stefan Mengel, and Friedrich Slivovsky. "On Compiling CNFs into Structured Deterministic DNNFs." In: *Theory and Applications of Satisfiability Testing – SAT 2015 – 18th International Conference, Austin, TX, USA, September 24–27, 2015, Proceedings*. Ed. by Marijn Heule and Sean A. Weaver. Vol. 9340. Lecture Notes in Computer Science. Springer, 2015, pp. 199–214. DOI: 10.1007/978-3-319-24318-4_15.

[32] Jörg Brauer, Andy King, and Jael Kriener. "Existential Quantification as Incremental SAT." In: *Computer Aided Verification – 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14–20, 2011. Proceedings*. Ed. by Ganesh Gopalakrishnan and Shaz Qadeer. Vol. 6806. Lecture Notes in Computer Science. Springer, 2011, pp. 191–207. DOI: 10.1007/978-3-642-22110-1_17.

[33] Martin Bromberger. "Decision Procedures for Linear Arithmetic. (Quelques procédures de décision pour l'arithmétique linéaire)." PhD thesis. Saarland University, Saarbrücken, Germany, 2019. URL: https://tel.archives-ouvertes.fr/tel-02427371.

[34] Martin Bromberger, Mathias Fleury, Simon Schwarz, and Christoph Weidenbach. "SPASS-SATT – A CDCL(LA) Solver." In: *Automated Deduction – CADE 27 – 27th International Conference on Automated Deduction, Natal, Brazil, August 27–30, 2019, Proceedings*. Ed. by Pascal Fontaine. Vol. 11716. Lecture Notes in Computer Science. Springer, 2019, pp. 111–122. DOI: 10.1007/978-3-030-29436-6_7.

[35] Robert Brummayer, Florian Lonsing, and Armin Biere. "Automated Testing and Debugging of SAT and QBF Solvers." In: *Theory and Applications of Satisfiability Testing – SAT 2010, 13th International Conference, SAT 2010, Edinburgh, UK, July 11–14, 2010. Proceedings*. Ed. by Ofer Strichman and Ste-

fan Szeider. Vol. 6175. Lecture Notes in Computer Science. Springer, 2010, pp. 44–57. DOI: 10.1007/978-3-642-14186-7_6.

[36] Jan Burchard, Dominik Erb, and Bernd Becker. “Characterization of Possibly Detected Faults by Accurately Computing their Detection Probability.” In: *2018 Design, Automation & Test in Europe Conference & Exhibition, DATE 2018, Dresden, Germany, March 19–23, 2018*. Ed. by Jan Madsen and Ayse K. Coskun. IEEE, 2018, pp. 385–390. DOI: 10.23919/DATE.2018.8342040.

[37] Jan Burchard, Tobias Schubert, and Bernd Becker. “Laissez-Faire Caching for Parallel #SAT Solving.” In: *Theory and Applications of Satisfiability Testing – SAT 2015 – 18th International Conference, Austin, TX, USA, September 24–27, 2015, Proceedings*. Ed. by Marijn Heule and Sean A. Weaver. Vol. 9340. Lecture Notes in Computer Science. Springer, 2015, pp. 46–61. DOI: 10.1007/978-3-319-24318-4_5.

[38] Jan Burchard, Tobias Schubert, and Bernd Becker. “Distributed Parallel #SAT Solving.” In: *2016 IEEE International Conference on Cluster Computing, CLUSTER 2016, Taipei, Taiwan, September 12–16, 2016*. IEEE Computer Society, 2016, pp. 326–335. DOI: 10.1109/CLUSTER.2016.20.

[39] Marco Cadoli and Francesco M. Donini. “A Survey on Knowledge Compilation.” In: *AI Communications* 10.3–4 (1997), pp. 137–150.

[40] Marco Cadoli, Francesco M. Donini, Paolo Liberatore, and Marco Schaerf. “Space Efficiency of Propositional Knowledge Representation Formalisms.” In: *Journal of Artificial Intelligence Research (JAIR)* 13 (2000), pp. 1–31. DOI: 10.1613/jair.664.

[41] Florent Capelli. “Understanding the complexity of #SAT using knowledge compilation.” In: *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20–23, 2017*. IEEE Computer Society, 2017, pp. 1–10. DOI: 10.1109/LICS.2017.8005121.

[42] Florent Capelli, Arnaud Durand, and Stefan Mengel. “Hypergraph Acyclicity and Propositional Model Counting.” In: *Theory and Applications of Satisfiability Testing – SAT 2014 – 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14–17, 2014. Proceedings*. Ed. by Carsten Sinz and Uwe Egly. Vol. 8561. Lecture Notes in Computer Science. Springer, 2014, pp. 399–414. DOI: 10.1007/978-3-319-09284-3_29.

[43] Supratik Chakraborty, Daniel J. Fremont, Kuldeep S. Meel, Sanjit A. Seshia, and Moshe Y. Vardi. “Distribution-Aware Sampling and Weighted Model Counting for SAT.” In: *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27–31, 2014, Québec City, Québec, Canada*. Ed. by Carla E. Brodley and Peter Stone. AAAI Press, 2014, pp. 1722–1730. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8364.

[44] Supratik Chakraborty, Dror Fried, Kuldeep S. Meel, and Moshe Y. Vardi. “From Weighted to Unweighted Model Counting.” In: *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25–31, 2015*. Ed. by Qiang Yang and Michael J. Wooldridge. AAAI Press, 2015, pp. 689–695. URL: http://ijcai.org/Abstract/15/103.

[45] Supratik Chakraborty, Kuldeep S. Meel, and Moshe Y. Vardi. "A Scalable Approximate Model Counter." In: *Principles and Practice of Constraint Programming – 19th International Conference, CP 2013, Uppsala, Sweden, September 16–20, 2013. Proceedings*. Ed. by Christian Schulte. Vol. 8124. Lecture Notes in Computer Science. Springer, 2013, pp. 200–216. DOI: 10.1007/978-3-642-40627-0_18.

[46] Supratik Chakraborty, Kuldeep S. Meel, and Moshe Y. Vardi. "Algorithmic Improvements in Approximate Counting for Probabilistic Inference: From Linear to Logarithmic SAT Calls." In: *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9–15 July 2016*. Ed. by Subbarao Kambhampati. IJCAI/AAAI Press, 2016, pp. 3569–3576. URL: http://www.ijcai.org/Abstract/16/503.

[47] Supratik Chakraborty, Kuldeep S. Meel, and Moshe Y. Vardi. "Approximate Model Counting." In: *Handbook of Satisfiability*. Ed. by Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh. Vol. 336. Frontiers in Artificial Intelligence and Applications. IOS Press, 2021, pp. 1015–1045. DOI: 10.3233/FAIA201010.

[48] Mark Chavira and Adnan Darwiche. "On probabilistic inference by weighted model counting." In: *Artificial Intelligence* 172.6–7 (2008), pp. 772–799.

[49] Christopher Condrat and Priyank Kalla. "A Gröbner Basis Approach to CNF-Formulae Preprocessing." In: *Tools and Algorithms for the Construction and Analysis of Systems, 13th International Conference, TACAS 2007, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2007 Braga, Portugal, March 24–April 1, 2007, Proceedings*. Ed. by Orna Grumberg and Michael Huth. Vol. 4424. Lecture Notes in Computer Science. Springer, 2007, pp. 618–631. DOI: 10.1007/978-3-540-71209-1_48.

[50] Bruno Courcelle and Stephan Olariu. "Upper bounds to the clique width of graphs." In: *Discrete Applied Mathematics* 101.1–3 (2000), pp. 77–114. DOI: 10.1016/S0166-218X(99)00184-5.

[51] Adnan Darwiche. "Compiling Knowledge into Decomposable Negation Normal Form." In: *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99, Stockholm, Sweden, July 31–August 6, 1999. 2 Vols., 1450 pages*. Ed. by Thomas Dean. Morgan Kaufmann, 1999, pp. 284–289. URL: http://ijcai.org/Proceedings/99-1/Papers/042.pdf.

[52] Adnan Darwiche. "Decomposable Negation Normal Form." In: *Journal of the ACM* 48.4 (2001), pp. 608–647. DOI: 10.1145/502090.502091.

[53] Adnan Darwiche. "On the Tractable Counting of Theory Models and its Application to Truth Maintenance and Belief Revision." In: *Journal of Applied Non-Classical Logics* 11.1–2 (2001), pp. 11–34. DOI: 10.3166/jancl.11.11-34.

[54] Adnan Darwiche. "A Compiler for Deterministic, Decomposable Negation Normal Form." In: *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence, July 28–August 1, 2002, Edmonton, Alberta, Canada*. Ed. by Rina Dechter, Michael J. Kearns, and Richard S. Sutton. AAAI Press / The MIT Press, 2002, pp. 627–634. URL: http://www.aaai.org/Library/AAAI/2002/aaai02-094.php.

[55] Adnan Darwiche. “New Advances in Compiling CNF into Decomposable Negation Normal Form.” In: *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, including Prestigious Applicants of Intelligent Systems, PAIS 2004, Valencia, Spain, August 22–27, 2004*. Ed. by Ramón López de Mántaras and Lorenza Saitta. IOS Press, 2004, pp. 328–332.

[56] Adnan Darwiche. “SDD: A New Canonical Representation of Propositional Knowledge Bases.” In: *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16–22, 2011*. Ed. by Toby Walsh. IJCAI/AAAI, 2011, pp. 819–826. DOI: 10.5591/978-1-57735-516-8/IJCAI11-143.

[57] Adnan Darwiche and KnotPipatsrisawat. “CompleteAlgorithms.” In: *Handbook of Satisfiability*. Ed. by Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh. Vol. 185. Frontiers in Artificial Intelligence and Applications. IOS Press, 2009, pp. 99–130. DOI: 10.3233/978-1-58603-929-5-99.

[58] Adnan Darwiche and KnotPipatsrisawat. “CompleteAlgorithms.” In: *Handbook of Satisfiability*. Ed. by Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh. Vol. 336. Frontiers in Artificial Intelligence and Applications. IOS Press, 2021, pp. 101–132. DOI: 10.3233/FAIA200986.

[59] Adnan Darwiche and Pierre Marquis. “A Knowledge Compilation Map.” In: *Journal of Artificial Intelligence Research (JAIR)* 17 (2002), pp. 229–264. DOI: 10.1613/jair.989.

[60] Martin Davis, George Logemann, and Donald W. Loveland. “A Machine Program for Theorem-Proving.” In: *Communications of the ACM* 5.7 (1962), pp. 394–397. DOI: 10.1145/368273.368557.

[61] Martin Davis and Hilary Putnam. “A Computing Procedure for Quantification Theory.” In: *Journal of the ACM* 7.3 (1960), pp. 201–215. DOI: 10.1145/321033.321034.

[62] Carmel Domshlak and Jörg Hoffmann. “Probabilistic Planning via Heuristic Forward Search and Weighted Model Counting.” In: *Journal of Artificial Intelligence Research (JAIR)* 30 (2007), pp. 565–620. DOI: 10.1613/jair.2289.

[63] Jeffrey M. Dudek, Vu H. N. Phan, and Moshe Y. Vardi. “DPMC: Weighted Model Counting by Dynamic Programming on Project-Join Trees.” In: *Principles and Practice of Constraint Programming – 26th International Conference, CP 2020, Louvain-la-Neuve, Belgium, September 7–11, 2020, Proceedings*. Ed. by Helmut Simonis. Vol. 12333. Lecture Notes in Computer Science. Springer, 2020, pp. 211–230. DOI: 10.1007/978-3-030-58475-7_13.

[64] Jeffrey M. Dudek, Vu Phan, and Moshe Y. Vardi. “ADDMC: Weighted Model Counting with Algebraic Decision Diagrams.” In: *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7–12, 2020*. AAAI Press, 2020, pp. 1468–1476. URL: https://aaai.org/ojs/index.php/AAAI/article/view/5505.

[65] Niklas Eén and Armin Biere. "Effective Preprocessing in SAT Through Variable and Clause Elimination." In: *Theory and Applications of Satisfiability Testing, 8th International Conference, SAT 2005, St. Andrews, UK, June 19–23, 2005, Proceedings*. Ed. by Fahiem Bacchus and Toby Walsh. Vol. 3569. Lecture Notes in Computer Science. Springer, 2005, pp. 61–75. DOI: 10.1007/11499107_5.

[66] Niklas Eén, Alan Mishchenko, and Niklas Sörensson. "Applying Logic Synthesis for Speeding Up SAT." In: *Theory and Applications of Satisfiability Testing – SAT 2007 – 10th International Conference, Lisbon, Portugal, May 28–31, 2007, Proceedings*. Ed. by João Marques-Silva and Karem A. Sakallah. Vol. 4501. Lecture Notes in Computer Science. Springer, 2007, pp. 272–286. DOI: 10.1007/978-3-540-72788-0_26.

[67] Niklas Eén and Niklas Sörensson. "Temporal induction by incremental SAT solving." In: *Electron. Notes Theor. Comput. Sci.* 89.4 (2003), pp. 543–560. DOI: 10.1016/S1571-0661(05)82542-3.

[68] Stefano Ermon, Carla P. Gomes, Ashish Sabharwal, and Bart Selman. "Low-density Parity Constraints for Hashing-Based Discrete Integration." In: *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21–26 June 2014*. Vol. 32. JMLR Workshop and Conference Proceedings. JMLR.org, 2014, pp. 271–279. URL: http://proceedings.mlr.press/v32/ermon14.html.

[69] Stefano Ermon, Carla P. Gomes, and Bart Selman. "Uniform Solution Sampling Using a Constraint Solver As an Oracle." In: *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence, Catalina Island, CA, USA, August 14–18, 2012*. Ed. by Nando de Freitas and Kevin P. Murphy. AUAI Press, 2012, pp. 255–264. URL: https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1&smnu=2&article_id=2288&proceeding_id=28.

[70] Hélène Fargier and Jérôme Mengin. "A Knowledge Compilation Map for Conditional Preference Statements-based Languages." In: *AAMAS '21: 20th International Conference on Autonomous Agents and Multiagent Systems, Virtual Event, United Kingdom, May 3-7, 2021*. Ed. by Frank Dignum, Alessio Lomuscio, Ulle Endriss, and Ann Nowé. ACM, 2021, pp. 492–500. URL: https://dl.acm.org/doi/10.5555/3463952.3464014.

[71] Katalin Fazekas, Armin Biere, and Christoph Scholl. "Incremental Inprocessing in SAT Solving." In: *Theory and Applications of Satisfiability Testing – SAT 2019 – 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9–12, 2019, Proceedings*. Ed. by Mikolás Janota and Inês Lynce. Vol. 11628. Lecture Notes in Computer Science. Springer, 2019, pp. 136–154. DOI: 10.1007/978-3-030-24258-9_9.

[72] Katalin Fazekas, Martina Seidl, and Armin Biere. "A Duality-Aware Calculus for Quantified Boolean Formulas." In: *18th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2016, Timisoara, Romania, September 24–27, 2016*. Ed. by James H. Davenport, Viorel Negru, Tetsuo Ida, Tudor Jebelean, Dana Petcu, Stephen M. Watt, and Daniela Zaharie. IEEE, 2016, pp. 181–186. DOI: 10.1109/SYNASC.2016.038.

[73] Linus Feiten, Matthias Sauer, Tobias Schubert, Alexander Czutro, Eberhard Böhl, Ilia Polian, and Bernd Becker. "#SAT-based Vulnerability Analysis of Security Components – A Case Study." In: *2012 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems, DFT 2012, Austin, TX, USA, October 3–5, 2012*. IEEE Computer Society, 2012, pp. 49–54. DOI: 10.1109/DFT.2012.6378198.

[74] Linus Feiten, Matthias Sauer, Tobias Schubert, Victor Tomashevich, Ilia Polian, and Bernd Becker. "Formal Vulnerability Analysis of Security Components." In: *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 34.8 (2015), pp. 1358–1369. DOI: 10.1109/TCAD.2015.2448687.

[75] Johannes Klaus Fichte, Markus Hecher, Michael Morak, and Stefan Woltran. "Exploiting Treewidth for Projected Model Counting and Its Limits." In: *Theory and Applications of Satisfiability Testing – SAT 2018 – 21st International Conference, SAT 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9–12, 2018, Proceedings*. Ed. by Olaf Beyersdorff and Christoph M. Wintersteiger. Vol. 10929. Lecture Notes in Computer Science. Springer, 2018, pp. 165–184. DOI: 10.1007/978-3-319-94144-8_11.

[76] Johannes Klaus Fichte, Markus Hecher, Stefan Woltran, and Markus Zisser. "Weighted Model Counting on the GPU by Exploiting Small Treewidth." In: *26th Annual European Symposium on Algorithms, ESA 2018, August 20–22, 2018, Helsinki, Finland*. Ed. by Yossi Azar, Hannah Bast, and Grzegorz Herman. Vol. 112. LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018, 28:1–28:16. DOI: 10.4230/LIPIcs.ESA.2018.28.

[77] Eldar Fischer, Johann A. Makowsky, and Elena V. Ravve. "Counting truth assignments of formulas of bounded tree-width or clique-width." In: *Discrete Applied Mathematics* 156.4 (2008), pp. 511–529. DOI: 10.1016/j.dam.2006.06.020.

[78] Mathias Fleury. "Formalisation of ground inference systems in a proof assistant." MA thesis. École normale supérieure de Rennes, 2015. URL: https://www.mpi-inf.mpg.de/fileadmin/inf/rg1/Documents/fleury_master_thesis.pdf.

[79] Anton Fuxjaeger and Vaishak Belle. "Scaling up Probabilistic Inference in Linear and Non-linear Hybrid Domains by Leveraging Knowledge Compilation." In: *Proceedings of the 12th International Conference on Agents and Artificial Intelligence, ICAART 2020, Volume 2, Valletta, Malta, February 22–24, 2020*. Ed. by Ana Paula Rocha, Luc Steels, and H. Jaap van den Herik. SCITEPRESS, 2020, pp. 347–355. DOI: 10.5220/0008896003470355.

[80] José Angel Galindo, Mathieu Acher, Juan Manuel Tirado, Cristian Vidal, Benoit Baudry, and David Benavides. "Exploiting the Enumeration of all Feature Model Configurations: A New Perspective with Distributed Computing." In: *Proceedings of the 20th International Systems and Software Product Line Conference, SPLC 2016, Beijing, China, September 16–23, 2016*. Ed. by Hong Mei. ACM, 2016, pp. 74–78. DOI: 10.1145/2934466.2934478.

[81] Robert Ganian and Stefan Szeider. "Community Structure Inspired Algorithms for SAT and #SAT." In: *Theory and Applications of Satisfiability Testing – SAT 2015 – 18th International Conference, Austin, TX, USA, September 24–27, 2015, Proceedings*. Ed. by Marijn Heule and Sean A. Weaver. Vol. 9340.

Lecture Notes in Computer Science. Springer, 2015, pp. 223–237. DOI: 10.1007/978-3-319-24318-4_17.

[82] Robert Ganian and Stefan Szeider. "New Width Parameters for Model Counting." In: *Theory and Applications of Satisfiability Testing – SAT 2017 – 20th International Conference, Melbourne, VIC, Australia, August 28–September 1, 2017, Proceedings*. Ed. by Serge Gaspers and Toby Walsh. Vol. 10491. Lecture Notes in Computer Science. Springer, 2017, pp. 38–52. DOI: 10.1007/978-3-319-66263-3_3.

[83] Martin Gebser, Benjamin Kaufmann, André Neumann, and Torsten Schaub. "*clasp*: A Conflict-Driven Answer Set Solver." In: *Logic Programming and Nonmonotonic Reasoning, 9th International Conference, LPNMR 2007, Tempe, AZ, USA, May 15–17, 2007, Proceedings*. Ed. by Chitta Baral, Gerhard Brewka, and John S. Schlipf. Vol. 4483. Lecture Notes in Computer Science. Springer, 2007, pp. 260–265. DOI: 10.1007/978-3-540-72200-7_23.

[84] Martin Gebser, Benjamin Kaufmann, and Torsten Schaub. "Solution Enumeration for Projected Boolean Search Problems." In: *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 6th International Conference, CPAIOR 2009, Pittsburgh, PA, USA, May 27–31, 2009, Proceedings*. Ed. by Willem Jan van Hoeve and John N. Hooker. Vol. 5547. Lecture Notes in Computer Science. Springer, 2009, pp. 71–86. DOI: 10.1007/978-3-642-01929-6_7.

[85] Vibhav Gogate and Rina Dechter. "Approximate Counting by Sampling the Backtrack-free Search Space." In: *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22–26, 2007, Vancouver, British Columbia, Canada*. AAAI Press, 2007, pp. 198–203. URL: http://www.aaai.org/Library/AAAI/2007/aaai07-030.php.

[86] Carla P. Gomes, Jörg Hoffmann, Ashish Sabharwal, and Bart Selman. "From Sampling to Model Counting." In: *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6–12, 2007*. Ed. by Manuela M. Veloso. 2007, pp. 2293–2299. URL: http://ijcai.org/Proceedings/07/Papers/369.pdf.

[87] Carla P. Gomes, Jörg Hoffmann, Ashish Sabharwal, and Bart Selman. "Short XORs for Model Counting: From Theory to Practice." In: *Theory and Applications of Satisfiability Testing – SAT 2007 – 10th International Conference, Lisbon, Portugal, May 28–31, 2007, Proceedings*. Ed. by João Marques-Silva and Karem A. Sakallah. Vol. 4501. Lecture Notes in Computer Science. Springer, 2007, pp. 100–106. DOI: 10.1007/978-3-540-72788-0_13.

[88] Carla P. Gomes, Ashish Sabharwal, and Bart Selman. "Model Counting: A New Strategy for Obtaining Good Bounds." In: *Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, July 16–20, 2006, Boston, Massachusetts, USA*. AAAI Press, 2006, pp. 54–61. URL: http://www.aaai.org/Library/AAAI/2006/aaai06-009.php.

[89] Carla P. Gomes, Ashish Sabharwal, and Bart Selman. "Model Counting." In: *Handbook of Satisfiability*. Ed. by Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh. Vol. 185. Frontiers in Artificial Intelligence and Applications. IOS Press, 2009, pp. 633–654. DOI: 10.3233/978-1-58603-929-5-633.

[90] Carla P. Gomes, Ashish Sabharwal, and Bart Selman. "Model Counting." In: *Handbook of Satisfiability*. Ed. by Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh. Vol. 336. Frontiers in Artificial Intelligence and Applications. IOS Press, 2021, pp. 993–1014. DOI: 10.3233/FAIA201009.

[91] Alexandra Goultiaeva and Fahiem Bacchus. "Exploiting QBF Duality on a Circuit Representation." In: *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11–15, 2010*. Ed. by Maria Fox and David Poole. AAAI Press, 2010. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1791.

[92] Alexandra Goultiaeva and Fahiem Bacchus. "Off the Trail: Re-examining the CDCL Algorithm." In: *Theory and Applications of Satisfiability Testing – SAT 2012 – 15th International Conference, Trento, Italy, June 17–20, 2012. Proceedings*. Ed. by Alessandro Cimatti and Roberto Sebastiani. Vol. 7317. Lecture Notes in Computer Science. Springer, 2012, pp. 30–43. DOI: 10.1007/978-3-642-31612-8_4.

[93] 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 13, Grenoble, France, March 18–22, 2013*. Ed. by Enrico Macii. EDA Consortium San Jose, CA, USA / ACM DL, 2013, pp. 811–814. DOI: 10.7873/DATE.2013.172.

[94] Orna Grumberg, Assaf Schuster, and Avi Yadgar. "Memory Efficient All-Solutions SAT Solver and Its Application for Reachability Analysis." In: *Formal Methods in Computer-Aided Design, 5th International Conference, FMCAD 2004, Austin, Texas, USA, November 15–17, 2004, Proceedings*. Ed. by Alan J. Hu and Andrew K. Martin. Vol. 3312. Lecture Notes in Computer Science. Springer, 2004, pp. 275–289. DOI: 10.1007/978-3-540-30494-4_20.

[95] Aarti Gupta, Zijiang Yang, Pranav Ashar, and Anubhav Gupta. "SAT-Based Image Computation with Application in Reachability Analysis." In: *Formal Methods in Computer-Aided Design, Third International Conference, FMCAD 2000, Austin, Texas, USA, November 1–3, 2000, Proceedings*. Ed. by Warren A. Hunt Jr. and Steven D. Johnson. Vol. 1954. Lecture Notes in Computer Science. Springer, 2000, pp. 354–371. DOI: 10.1007/3-540-40922-X_22.

[96] Rui Henriques, Inês Lynce, and Vasco M. Manquinho. "On When and How to use SAT to Mine Frequent Itemsets." In: *CoRR* abs/1207.6253 (2012). arXiv: 1207.6253. URL: http://arxiv.org/abs/1207.6253.

[97] Steffen Hölldobler, Norbert Manthey, Tobias Philipp, and Peter Steinke. "Generic CDCL – A Formalization of Modern Propositional Satisfiability Solvers." In: *POS-14. Fifth Pragmatics of SAT workshop, a workshop of the SAT 2014 conference, part of FLoC 2014 during the Vienna Summer of Logic, July 13, 2014, Vienna, Austria*. Ed. by Daniel Le Berre. Vol. 27. EPiC Series in Computing. EasyChair, 2014, pp. 89–102. URL: https://easychair.org/publications/paper/nTn.

[98] John N. Hooker. "Solving the incremental satisfiability problem." In: *J. Log. Program.* 15.1–2 (1993), pp. 177–186. DOI: 10.1016/0743-1066(93)90018-C.

[99] Jinbo Huang and Adnan Darwiche. "DPLL with a Trace: From SAT to Knowledge Compilation." In: *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30–August 5, 2005*. Ed. by Leslie Pack Kaelbling and Alessandro Saffiotti. Professional Book Center, 2005, pp. 156–162. URL: http://ijcai.org/Proceedings/05/Papers/0876.pdf.

[100] Jinbo Huang and Adnan Darwiche. "The Language of Search." In: *Journal of Artificial Intelligence Research (JAIR)* 29 (2007), pp. 191–219. DOI: 10.1613/jair.2097.

[101] Markus Iser, Carsten Sinz, and Mana Taghdiri. "Minimizing Models for Tseitin-Encoded SAT Instances." In: *Theory and Applications of Satisfiability Testing – SAT 2013 – 16th International Conference, Helsinki, Finland, July 8–12, 2013. Proceedings*. Ed. by Matti Järvisalo and Allen Van Gelder. Vol. 7962. Lecture Notes in Computer Science. Springer, 2013, pp. 224–232. DOI: 10.1007/978-3-642-39071-5_17.

[102] Mark Jerrum, Leslie G. Valiant, and Vijay V. Vazirani. "Random Generation of Combinatorial Structures from a Uniform Distribution." In: *Theor. Comput. Sci.* 43 (1986), pp. 169–188. DOI: 10.1016/0304-3975(86)90174-X.

[103] HoonSang Jin, HyoJung Han, and Fabio Somenzi. "Efficient Conflict Analysis for Finding All Satisfying Assignments of a Boolean Circuit." In: *Tools and Algorithms for the Construction and Analysis of Systems, 11th International Conference, TACAS 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4–8, 2005, Proceedings*. Ed. by Nicolas Halbwachs and Lenore D. Zuck. Vol. 3440. Lecture Notes in Computer Science. Springer, 2005, pp. 287–300. DOI: 10.1007/978-3-540-31980-1_19.

[104] Vladimir Klebanov, Norbert Manthey, and Christian J. Muise. "SAT-Based Analysis and Quantification of Information Flow in Programs." In: *Quantitative Evaluation of Systems – 10th International Conference, QEST 2013, Buenos Aires, Argentina, August 27–30, 2013. Proceedings*. Ed. by Kaustubh R. Joshi, Markus Siegle, Mariëlle Stoelinga, and Pedro R. D'Argenio. Vol. 8054. Lecture Notes in Computer Science. Springer, 2013, pp. 177–192. DOI: 10.1007/978-3-642-40196-1_16.

[105] Vladimir Klebanov, Alexander Weigl, and Jörg Weisbarth. "Sound Probabilistic #SAT with Projection." In: *Proceedings 14th International Workshop Quantitative Aspects of Programming Languages and Systems, QAPL 2016, Eindhoven, The Netherlands, April 2–3, 2016*. Ed. by Mirco Tribastone and Herbert Wiklicky. Vol. 227. EPTCS. 2016, pp. 15–29. DOI: 10.4204/EPTCS.227.2.

[106] Stefan Kölbl, Gregor Leander, and Tyge Tiessen. "Observations on the SIMON Block Cipher Family." In: *Advances in Cryptology – CRYPTO 2015 – 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16–20, 2015, Proceedings, Part I*. Ed. by Rosario Gennaro and Matthew Robshaw. Vol. 9215. Lecture Notes in Computer Science. Springer, 2015, pp. 161–185. DOI: 10.1007/978-3-662-47989-6_8.

[107] Timothy Kopp, Parag Singla, and Henry A. Kautz. "Toward Caching Symmetrical Subtheories for Weighted Model Counting." In: *Beyond NP, Papers from the 2016 AAAI Workshop, Phoenix, Arizona, USA, February 12, 2016*. Ed. by Adnan Darwiche. Vol. WS-16-05. AAAI Workshops. AAAI Press, 2016.

URL: http://www.aaai.org/ocs/index.php/WS/AAAIW16/paper/view/12684.

[108] Frédéric Koriche, Jean-Marie Lagniez, Pierre Marquis, and Samuel Thomas. "Knowledge Compilation for Model Counting: Affine Decision Trees." In: *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3–9, 2013*. Ed. by Francesca Rossi. IJCAI/AAAI, 2013, pp. 947–953. URL: http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6574.

[109] Lukas Kroc, Ashish Sabharwal, and Bart Selman. "Leveraging Belief Propagation, Backtrack Search, and Statistics for Model Counting." In: *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 5th International Conference, CPAIOR 2008, Paris, France, May 20–23, 2008, Proceedings*. Ed. by Laurent Perron and Michael A. Trick. Vol. 5015. Lecture Notes in Computer Science. Springer, 2008, pp. 127–141. DOI: 10.1007/978-3-540-68155-7_12.

[110] Andreas Kübler, Christoph Zengler, and Wolfgang Küchlin. "Model Counting in Product Configuration." In: *Proceedings First International Workshop on Logics for Component Configuration, LoCoCo 2010, Edinburgh, UK, 10th July 2010*. Ed. by Inês Lynce and Ralf Treinen. Vol. 29. EPTCS. 2010, pp. 44–53. DOI: 10.4204/EPTCS.29.5.

[111] Jonathan Kuck, Tri Dao, Shenjia Zhao, Burak Bartan, Ashish Sabharwal, and Stefano Ermon. "Adaptive Hashing for Model Counting." In: *Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI 2019, Tel Aviv, Israel, July 22–25, 2019*. Ed. by Amir Globerson and Ricardo Silva. Vol. 115. Proceedings of Machine Learning Research. AUAI Press, 2019, pp. 271–280. URL: http://proceedings.mlr.press/v115/kuck20a.html.

[112] T.K. Satish Kumar. "A Model Counting Characterization of Diagnoses." In: *Proceedings of the Thirteenth International Workshop on Principles of Diagnosis, DX-2002*. 2002, pp. 70–76.

[113] Jean-Marie Lagniez, Emmanuel Lonca, and Pierre Marquis. "Improving Model Counting by Leveraging Definability." In: *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9–15 July 2016*. Ed. by Subbarao Kambhampati. IJCAI/AAAI Press, 2016, pp. 751–757. URL: http://www.ijcai.org/Abstract/16/112.

[114] Jean-Marie Lagniez and Pierre Marquis. "An Improved Decision-DNNF Compiler." In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19–25, 2017*. Ed. by Carles Sierra. ijcai.org, 2017, pp. 667–673. DOI: 10.24963/ijcai.2017/93.

[115] Jean-Marie Lagniez and Pierre Marquis. "On Preprocessing Techniques and Their Impact on Propositional Model Counting." In: *J. Autom. Reason.* 58.4 (2017), pp. 413–481. DOI: 10.1007/s10817-016-9370-8.

[116] Jean-Marie Lagniez and Pierre Marquis. "A Recursive Algorithm for Projected Model Counting." In: *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January*

27–February 1, 2019. AAAI Press, 2019, pp. 1536–1543. DOI: 10.1609/aaai.v33i01.33011536.

[117] Jean-Marie Lagniez, Pierre Marquis, and Nicolas Szczepanski. "DMC: A Distributed Model Counter." In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13–19, 2018, Stockholm, Sweden*. Ed. by Jérôme Lang. ijcai.org, 2018, pp. 1331–1338. DOI: 10.24963/ijcai.2018/185.

[118] Shuvendu K. Lahiri, Robert Nieuwenhuis, and Albert Oliveras. "SMT Techniques for Fast Predicate Abstraction." In: *Computer Aided Verification, 18th International Conference, CAV 2006, Seattle, WA, USA, August 17–20, 2006, Proceedings*. Ed. by Thomas Ball and Robert B. Jones. Vol. 4144. Lecture Notes in Computer Science. Springer, 2006, pp. 424–437. DOI: 10.1007/11817963_39.

[119] Bin Li, Michael S. Hsiao, and Shuo Sheng. "A Novel SAT All-Solutions Solver for Efficient Preimage Computation." In: *2004 Design, Automation and Test in Europe Conference and Exposition (DATE 2004), 16–20 February 2004, Paris, France*. IEEE Computer Society, 2004, pp. 272–279. DOI: 10.1109/DATE.2004.1268860.

[120] Wei Li, Peter van Beek, and Pascal Poupart. "Performing Incremental Bayesian Inference by Dynamic Model Counting." In: *Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, July 16–20, 2006, Boston, Massachusetts, USA*. AAAI Press, 2006, pp. 1173–1179. URL: http://www.aaai.org/Library/AAAI/2006/aaai06-184.php.

[121] Jiajing Ling, Kushagra Chandak, and Akshat Kumar. "Integrating Knowledge Compilation with Reinforcement Learning for Routes." In: *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling, ICAPS 2021, Guangzhou, China (virtual), August 2–13, 2021*. Ed. by Susanne Biundo, Minh Do, Robert Goldman, Michael Katz, Qiang Yang, and Hankz Hankui Zhuo. AAAI Press, 2021, pp. 542–550. URL: https://ojs.aaai.org/index.php/ICAPS/article/view/16002.

[122] Florian Lonsing and Uwe Egly. "Incremental QBF Solving." In: *Principles and Practice of Constraint Programming – 20th International Conference, CP 2014, Lyon, France, September 8–12, 2014. Proceedings*. Ed. by Barry O'Sullivan. Vol. 8656. Lecture Notes in Computer Science. Springer, 2014, pp. 514–530. DOI: 10.1007/978-3-319-10428-7_38.

[123] Michael Makkai. "Admissible Sets and Infinitary Logic." In: *Handbook of Mathematical Logic*. Ed. by Jon Barwise. Vol. 90. Studies in Logic and the Foundations of Mathematics. North-Holland, 1977, pp. 233–281. DOI: 10.1016/S0049-237X(08)71103-0.

[124] Filip Marić and Predrag Janičić. "Formalization of Abstract State Transition Systems for SAT." In: *Logical Methods in Computer Science* 7.3 (2011). DOI: 10.2168/LMCS-7(3:19)2011.

[125] João P. Marques-Silva, Inês Lynce, and Sharad Malik. "Conflict-Driven Clause Learning SAT Solvers." In: *Handbook of Satisfiability*. Ed. by Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh. Vol. 185. Frontiers in Artificial Intelligence and Applications. IOS Press, 2009, pp. 131–153. DOI: 10.3233/978-1-58603-929-5-131.

[126] João P. Marques-Silva, Inês Lynce, and Sharad Malik. “Conflict-Driven Clause Learning SAT Solvers.” In: *Handbook of Satisfiability*. Ed. by Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh. Vol. 336. Frontiers in Artificial Intelligence and Applications. IOS Press, 2021, pp. 133–182. DOI: 10.3233/FAIA200987.

[127] João P. Marques-Silva and Karem A. Sakallah. “GRASP – A New Search Algorithm for Satisfiability.” In: *Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 1996, San Jose, CA, USA, November 10–14, 1996*. Ed. by Rob A. Rutenbar and Ralph H. J. M. Otten. IEEE Computer Society / ACM, 1996, pp. 220–227. DOI: 10.1109/ICCAD.1996.569607.

[128] João P. Marques-Silva and Karem A. Sakallah. “GRASP: A Search Algorithm for Propositional Satisfiability.” In: *IEEE Trans. Computers* 48.5 (1999), pp. 506–521. DOI: 10.1109/12.769433.

[129] João Marques-Silva. “Computing with SAT Oracles: Past, Present and Future.” In: *Sailing Routes in the World of Computation – 14th Conference on Computability in Europe, CiE 2018, Kiel, Germany, July 30–August 3, 2018, Proceedings*. Ed. by Florin Manea, Russell G. Miller, and Dirk Nowotka. Vol. 10936. Lecture Notes in Computer Science. Springer, 2018, pp. 264–276. DOI: 10.1007/978-3-319-94418-0_27.

[130] Robert Mateescu. *Treewidth in Industrial SAT Benchmarks*. Technical report MSR-TR-2011-22. 7 J J Thomson Avenue, Cambridge CB3 0FB, UK: Microsoft Research, 2011.

[131] Kenneth L. McMillan. “Applying SAT Methods in Unbounded Symbolic Model Checking.” In: *Computer Aided Verification, 14th International Conference, CAV 2002,Copenhagen, Denmark, July 27–31, 2002, Proceedings*. Ed. by Ed Brinksma and Kim Guldstrand Larsen. Vol. 2404. Lecture Notes in Computer Science. Springer, 2002, pp. 250–264. DOI: 10.1007/3-540-45657-0_19.

[132] Kenneth L. McMillan. “Interpolation and SAT-Based Model Checking.” In: *Computer Aided Verification, 15th International Conference, CAV 2003, Boulder, CO, USA, July 8–12, 2003, Proceedings*. Ed. by Warren A. Hunt Jr. and Fabio Somenzi. Vol. 2725. Lecture Notes in Computer Science. Springer, 2003, pp. 1–13. DOI: 10.1007/978-3-540-45069-6_1.

[133] Hakan Metin, Souheib Baarir, Maximilien Colange, and Fabrice Kordon. “CDCLSym: Introducing Effective Symmetry Breaking in SAT Solving.” In: *Tools and Algorithms for the Construction and Analysis of Systems – 24th International Conference, TACAS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14–20, 2018, Proceedings, Part I*. Ed. by Dirk Beyer and Marieke Huisman. Vol. 10805. Lecture Notes in Computer Science. Springer, 2018, pp. 99–114. DOI: 10.1007/978-3-319-89960-2_6.

[134] Peter Bro Miltersen, Jaikumar Radhakrishnan, and Ingo Wegener. “On converting CNF to DNF.” In: *Theor. Comput. Sci.* 347.1–2 (2005), pp. 325–335. DOI: 10.1016/j.tcs.2005.07.029.

[135] Shin-ichi Minato. “Fast Generation of Prime-Irredundant Covers from Binary Decision Diagrams.” In: *IEICE Trans. Fundamentals* E76-A.6 (1993), pp. 967–973.

[136] Shin-ichi Minato and Giovanni De Micheli. "Finding all simple disjunctive decompositions using irredundant sum-of-products forms." In: *Proceedings of the 1998 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 1998, San Jose, CA, USA, November 8-12, 1998*. Ed. by Hiroto Yasuura. ACM / IEEE Computer Society, 1998, pp. 111–117. DOI: 10.1145/288548.288586.

[137] Sibylle Möhle and Armin Biere. "Dualizing Projected Model Counting." In: *IEEE 30th International Conference on Tools with Artificial Intelligence, ICTAI 2018, 5–7 November 2018, Volos, Greece*. Ed. by Lefteri H. Tsoukalas, Éric Grégoire, and Miltiadis Alamaniotis. IEEE, 2018, pp. 702–709. DOI: 10.1109/ICTAI.2018.00111.

[138] Sibylle Möhle and Armin Biere. "Backing Backtracking." In: *Theory and Applications of Satisfiability Testing – SAT 2019 – 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9–12, 2019, Proceedings*. Ed. by Mikolás Janota and Inês Lynce. Vol. 11628. Lecture Notes in Computer Science. Springer, 2019, pp. 250–266. DOI: 10.1007/978-3-030-24258-9_18.

[139] Sibylle Möhle and Armin Biere. "Combining Conflict-Driven Clause Learning and Chronological Backtracking for Propositional Model Counting." In: *GCAI 2019. Proceedings of the 5th Global Conference on Artificial Intelligence, Bozen/Bolzano, Italy, 17–19 September 2019*. Ed. by Diego Calvanese and Luca Iocchi. Vol. 65. EPiC Series in Computing. EasyChair, 2019, pp. 113–126. DOI: 10.29007/vgg4.

[140] Sibylle Möhle, Roberto Sebastiani, and Armin Biere. "On Enumerating Short Projected Models." In: *CoRR* abs/2110.12924 (2021). arXiv: 2110.12924. URL: https://arxiv.org/abs/2110.12924.

[141] Sibylle Möhle, Roberto Sebastiani, and Armin Biere. "Four Flavors of Entailment." In: *Theory and Applications of Satisfiability Testing – SAT 2020 – 23rd International Conference, Alghero, Italy, July 3–10, 2020, Proceedings*. Ed. by Luca Pulina and Martina Seidl. Vol. 12178. Lecture Notes in Computer Science. Springer, 2020, pp. 62–71. DOI: 10.1007/978-3-030-51825-7_5.

[142] Paolo Morettin, Andrea Passerini, and Roberto Sebastiani. "Efficient Weighted Model Integration via SMT-Based Predicate Abstraction." In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19–25, 2017*. Ed. by Carles Sierra. ijcai.org, 2017, pp. 720–728. DOI: 10.24963/ijcai.2017/100.

[143] Paolo Morettin, Andrea Passerini, and Roberto Sebastiani. "Advanced SMT techniques for weighted model integration." In: *Artif. Intell.* 275 (2019), pp. 1–27. DOI: 10.1016/j.artint.2019.04.003.

[144] António José dos Reis Morgado and João Marques-Silva. *Algorithms for Propositional Model Enumeration and Counting*. Tech. rep. 39. INESC-ID, 2005.

[145] António Morgado and João P. Marques-Silva. "Good Learning and Implicit Model Enumeration." In: *17th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2005), 14–16 November 2005, Hong Kong, China*. IEEE Computer Society, 2005, pp. 131–136. DOI: 10.1109/ICTAI.2005.69.

[146] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. "Chaff: Engineering an Efficient SAT Solver." In: *Proceedings of the 38th Design Automation Conference, DAC 2001, Las Vegas, NV, USA, June 18–22, 2001*. ACM, 2001, pp. 530–535. DOI: 10.1145/378239.379017.

[147] Christian J. Muise, Sheila A. McIlraith, J. Christopher Beck, and Eric I. Hsu. "Fast d-DNNF Compilation with sharpSAT." In: *Abstraction, Reformulation, and Approximation, Papers from the 2010 AAAI Workshop, Atlanta, Georgia, USA, July 12, 2010*. Vol. WS-10-08. AAAI Workshops. AAAI, 2010. URL: http://aaai.org/ocs/index.php/WS/AAAIW10/paper/view/2069.

[148] Christian J. Muise, Sheila A. McIlraith, J. Christopher Beck, and Eric I. Hsu. "DSHARP: Fast d-DNNF Compilation with sharpSAT." In: *Advances in Artificial Intelligence – 25th Canadian Conference on Artificial Intelligence, Canadian AI 2012, Toronto, ON, Canada, May 28–30, 2012. Proceedings*. Ed. by Leila Kosseim and Diana Inkpen. Vol. 7310. Lecture Notes in Computer Science. Springer, 2012, pp. 356–361. DOI: 10.1007/978-3-642-30353-1_36.

[149] Alexander Nadel and Vadim Ryvchin. "Chronological Backtracking." In: *Theory and Applications of Satisfiability Testing – SAT 2018 – 21st International Conference, SAT 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9–12, 2018, Proceedings*. Ed. by Olaf Beyersdorff and Christoph M. Wintersteiger. Vol. 10929. Lecture Notes in Computer Science. Springer, 2018, pp. 111–121. DOI: 10.1007/978-3-319-94144-8_7.

[150] Alexander Nadel and Vadim Ryvchin. "Maple_LCM_Dist_ChronoBT: Featuring Chronological Backtracking." In: *Proceedings of SAT Competition 2018: Solver and Benchmark Descriptions*. Ed. by Marijn Heule, Matti Järvisalo, and Martin Suda. Vol. B-2018-1. Department of Computer Science Series of Publications B. University of Helsinki, 2018, p. 29. URL: https://helda.helsinki.fi/handle/10138/237063.

[151] Alexander Nadel, Vadim Ryvchin, and Ofer Strichman. "Preprocessing in Incremental SAT." In: *Theory and Applications of Satisfiability Testing – SAT 2012 – 15th International Conference, Trento, Italy, June 17–20, 2012. Proceedings*. Ed. by Alessandro Cimatti and Roberto Sebastiani. Vol. 7317. Lecture Notes in Computer Science. Springer, 2012, pp. 256–269. DOI: 10.1007/978-3-642-31612-8_20.

[152] Nina Narodytska, Aditya A. Shrotri, Kuldeep S. Meel, Alexey Ignatiev, and João Marques-Silva. "Assessing Heuristic Machine Learning Explanations with Model Counting." In: *Theory and Applications of Satisfiability Testing – SAT 2019 – 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9–12, 2019, Proceedings*. Ed. by Mikolás Janota and Inês Lynce. Vol. 11628. Lecture Notes in Computer Science. Springer, 2019, pp. 267–278. DOI: 10.1007/978-3-030-24258-9_19.

[153] Aina Niemetz, Mathias Preiner, and Armin Biere. "Turbo-charging Lemmas on demand with don't care reasoning." In: *Formal Methods in Computer-Aided Design, FMCAD 2014, Lausanne, Switzerland, October 21–24, 2014*. IEEE, 2014, pp. 179–186. DOI: 10.1109/FMCAD.2014.6987611.

[154] Aina Niemetz, Mathias Preiner, and Armin Biere. "Model-Based API Testing for SMT Solvers." In: *Proceedings of the 15th International Workshop on Satisfiability Modulo Theories, SMT 2017), affiliated with the 29th International Conference on Computer Aided Verification, CAV 2017, Heidelberg, Germany, July 24–28, 2017*. Ed. by Martin Brain and Liana Hadarean. 2017, 10 pages.

[155] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. "Solving SAT and SAT Modulo Theories: From an Abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T)." In: *J. ACM* 53.6 (2006), pp. 937–977. DOI: 10.1145/1217856.1217859.

[156] Chanseok Oh. "Between SAT and UNSAT: The Fundamental Difference in CDCL SAT." In: *Theory and Applications of Satisfiability Testing – SAT 2015 – 18th International Conference, Austin, TX, USA, September 24–27, 2015, Proceedings*. Ed. by Marijn Heule and Sean A. Weaver. Vol. 9340. Lecture Notes in Computer Science. Springer, 2015, pp. 307–323. DOI: 10.1007/978-3-319-24318-4_23.

[157] Chanseok Oh. "Improving SAT Solvers by Exploiting Empirical Characteristics of CDCL." PhD thesis. New York University, Department of Computer Science, 2016.

[158] Muhammad Osama, Anton Wijs, and Armin Biere. "SAT Solving with GPU Accelerated Inprocessing." In: *Tools and Algorithms for the Construction and Analysis of Systems – 27th International Conference, TACAS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27–April 1, 2021, Proceedings, Part I*. Ed. by Jan Friso Groote and Kim Guldstrand Larsen. Vol. 12651. Lecture Notes in Computer Science. Springer, 2021, pp. 133–151. DOI: 10.1007/978-3-030-72016-2_8.

[159] Umut Oztok and Adnan Darwiche. "On Compiling CNF into Decision-DNNF." In: *Principles and Practice of Constraint Programming – 20th International Conference, CP 2014, Lyon, France, September 8–12, 2014. Proceedings*. Ed. by Barry O'Sullivan. Vol. 8656. Lecture Notes in Computer Science. Springer, 2014, pp. 42–57. DOI: 10.1007/978-3-319-10428-7_7.

[160] Umut Oztok and Adnan Darwiche. "A Top-Down Compiler for Sentential Decision Diagrams." In: *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25–31, 2015*. Ed. by Qiang Yang and Michael J. Wooldridge. AAAI Press, 2015, pp. 3141–3148. URL: http://ijcai.org/Abstract/15/443.

[161] Umut Oztok and Adnan Darwiche. "An Exhaustive DPLL Algorithm for Model Counting." In: *Journal of Artificial Intelligence Research (JAIR)* 62 (2018), pp. 1–32. DOI: 10.1613/jair.1.11201.

[162] Héctor Palacios, Blai Bonet, Adnan Darwiche, and Hector Geffner. "Pruning Conformant Plans by Counting Models on Compiled d-DNNF Representations." In: *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS 2005), June 5–10 2005, Monterey, California, USA*. Ed. by Susanne Biundo, Karen L. Myers, and Kanna Rajan. AAAI, 2005, pp. 141–150. URL: http://www.aaai.org/Library/ICAPS/2005/icaps05-015.php.

[163] Knot Pipatsrisawat and Adnan Darwiche. "New Compilation Languages Based on Structured Decomposability." In: *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13–17, 2008*. Ed. by Dieter Fox and Carla P. Gomes. AAAI Press, 2008, pp. 517–522. URL: http://www.aaai.org/Library/AAAI/2008/aaai08-082.php.

[164] David A. Plaisted and Steven Greenbaum. "A Structure-Preserving Clause Form Translation." In: *J. Symb. Comput.* 2.3 (1986), pp. 293–304. DOI: 10.1016/S0747-7171(86)80028-1.

[165] Kavita Ravi and Fabio Somenzi. "Minimal Assignments for Bounded Model Checking." In: *Tools and Algorithms for the Construction and Analysis of Systems, 10th International Conference, TACAS 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, March 29–April 2, 2004, Proceedings*. Ed. by Kurt Jensen and Andreas Podelski. Vol. 2988. Lecture Notes in Computer Science. Springer, 2004, pp. 31–45. DOI: 10.1007/978-3-540-24730-2_3.

[166] John Alan Robinson. "A Machine-Oriented Logic Based on the Resolution Principle." In: *J. ACM* 12.1 (1965), pp. 23–41. DOI: 10.1145/321250.321253.

[167] Dan Roth. "On the Hardness of Approximate Reasoning." In: *Artif. Intell.* 82.1–2 (1996), pp. 273–302. DOI: 10.1016/0004-3702(94)00092-1.

[168] Marko Samer and Stefan Szeider. "Backdoor Sets of Quantified Boolean Formulas." In: *J. Autom. Reason.* 42.1 (2009), pp. 77–97. DOI: 10.1007/s10817-008-9114-5.

[169] Marko Samer and Stefan Szeider. "Fixed-Parameter Tractability." In: *Handbook of Satisfiability*. Ed. by Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh. Vol. 185. Frontiers in Artificial Intelligence and Applications. IOS Press, 2009, pp. 425–454. DOI: 10.3233/978-1-58603-929-5-425.

[170] Marko Samer and Stefan Szeider. "Algorithms for propositional model counting." In: *J. Discrete Algorithms* 8.1 (2010), pp. 50–64. DOI: 10.1016/j.jda.2009.06.002.

[171] Marko Samer and Stefan Szeider. "Fixed-Parameter Tractability." In: *Handbook of Satisfiability*. Ed. by Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh. Vol. 336. Frontiers in Artificial Intelligence and Applications. IOS Press, 2021, pp. 693–736. DOI: 10.3233/FAIA201000.

[172] Tian Sang, Fahiem Bacchus, Paul Beame, Henry A. Kautz, and Toniann Pitassi. "Combining Component Caching and Clause Learning for Effective Model Counting." In: *SAT 2004 – The Seventh International Conference on Theory and Applications of Satisfiability Testing, 10–13 May 2004, Vancouver, BC, Canada, Online Proceedings*. 2004. URL: http://www.satisfiability.org/SAT04/programme/21.pdf.

[173] Tian Sang, Paul Beame, and Henry A. Kautz. "Heuristics for Fast Exact Model Counting." In: *Theory and Applications of Satisfiability Testing, 8th International Conference, SAT 2005, St. Andrews, UK, June 19–23, 2005, Proceedings*. Ed. by Fahiem Bacchus and Toby Walsh. Vol. 3569. Lecture Notes in Computer Science. Springer, 2005, pp. 226–240. DOI: 10.1007/11499107_17.

[174] Tian Sang, Paul Beame, and Henry A. Kautz. "Performing Bayesian Inference by Weighted Model Counting." In: *Proceedings of the Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9–13, 2005, Pittsburgh, Pennsylvania, USA*. Ed. by Manuela M. Veloso and Subbarao Kambhampati. AAAI Press / The MIT Press, 2005, pp. 475–482. URL: http://www.aaai.org/Library/AAAI/2005/aaai05-075.php.

[175] Sergej Scheck, Alexandre Niveau, and Bruno Zanuttini. "Knowledge Compilation for Nondeterministic Action Languages." In: *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling, ICAPS 2021, Guangzhou, China (virtual), August 2–13, 2021*. Ed. by Susanne Biundo, Minh Do, Robert Goldman, Michael Katz, Qiang Yang, and Hankz Hankui Zhuo. AAAI Press, 2021, pp. 308–316. URL: https://ojs.aaai.org/index.php/ICAPS/article/view/15975.

[176] Roberto Sebastiani. "Lazy Satisfiability Modulo Theories." In: *Journal on Satisfiability, Boolean Modeling and Computation* 3.3–4 (2007), pp. 141–224. DOI: 10.3233/sat190034.

[177] Roberto Sebastiani. "Are You Satisfied by This Partial Assignment?" In: *CoRR* abs/2003.04225 (2020). arXiv: 2003.04225. URL: https://arxiv.org/abs/2003.04225.

[178] Bart Selman and Henry A. Kautz. "Knowledge Compilation and Theory Approximation." In: *J. ACM* 43.2 (1996), pp. 193–224. DOI: 10.1145/226643.226644.

[179] Shubham Sharma, Subhajit Roy, Mate Soos, and Kuldeep S. Meel. "GANAK: A Scalable Probabilistic Exact Model Counter." In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10–16, 2019*. Ed. by Sarit Kraus. ijcai.org, 2019, pp. 1169–1176. DOI: 10.24963/ijcai.2019/163.

[180] Shuo Sheng and Michael S. Hsiao. "Efficient Preimage Computation Using A Novel Success-Driven ATPG." In: *2003 Design, Automation and Test in Europe Conference and Exposition (DATE 2003), 3–7 March 2003, Munich, Germany*. IEEE Computer Society, 2003, pp. 10822–10827. DOI: 10.1109/DATE.2003.10125.

[181] Friedrich Slivovsky and Stefan Szeider. "Model Counting for Formulas of Bounded Clique-Width." In: *Algorithms and Computation – 24th International Symposium, ISAAC 2013, Hong Kong, China, December 16–18, 2013, Proceedings*. Ed. by Leizhen Cai, Siu-Wing Cheng, and Tak Wah Lam. Vol. 8283. Lecture Notes in Computer Science. Springer, 2013, pp. 677–687. DOI: 10.1007/978-3-642-45030-3_63.

[182] Friedrich Slivovsky and Stefan Szeider. "A Faster Algorithm for Propositional Model Counting Parameterized by Incidence Treewidth." In: *Theory and Applications of Satisfiability Testing – SAT 2020 – 23rd International Conference, Alghero, Italy, July 3–10, 2020, Proceedings*. Ed. by Luca Pulina and Martina Seidl. Vol. 12178. Lecture Notes in Computer Science. Springer, 2020, pp. 267–276. DOI: 10.1007/978-3-030-51825-7_19.

[183] Mate Soos and Kuldeep S. Meel. "BIRD: Engineering an Efficient CNF-XOR SAT Solver and Its Applications to Approximate Model Counting." In: *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27–February 1, 2019*. AAAI Press, 2019, pp. 1592–1599. DOI: 10.1609/aaai.v33i01.33011592.

[184] Ofer Strichman. "Tuning SAT Checkers for Bounded Model Checking." In: *Computer Aided Verification, 12th International Conference, CAV 2000, Chicago, IL, USA, July 15–19, 2000, Proceedings*. Ed. by E. Allen Emerson and A. Prasad Sistla. Vol. 1855. Lecture Notes in Computer Science. Springer, 2000, pp. 480–494. DOI: 10.1007/10722167_36.

[185] 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 2001, Livingston, Scotland, UK, September 4–7, 2001, Proceedings*. Ed. by Tiziana Margaria and Thomas F. Melham. Vol. 2144. Lecture Notes in Computer Science. Springer, 2001, pp. 58–70. DOI: 10.1007/3-540-44798-9_4.

[186] Sathiamoorthy Subbarayan, Lucas Bordeaux, and Youssef Hamadi. "Knowledge Compilation Properties of Tree-of-BDDs." In: *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22–26, 2007, Vancouver, British Columbia, Canada*. AAAI Press, 2007, pp. 502–507. URL: http://www.aaai.org/Library/AAAI/2007/aaai07-079.php.

[187] Allison Sullivan, Darko Marinov, and Sarfraz Khurshid. "Solution Enumeration Abstraction: A Modeling Idiom to Enhance a Lightweight Formal Method." In: *Formal Methods and Software Engineering – 21st International Conference on Formal Engineering Methods, ICFEM 2019, Shenzhen, China, November 5–9, 2019, Proceedings*. Ed. by Yamine Aït Ameur and Shengchao Qin. Vol. 11852. Lecture Notes in Computer Science. Springer, 2019, pp. 336–352. DOI: 10.1007/978-3-030-32409-4_21.

[188] Peter van der Tak, Antonio Ramos, and Marijn Heule. "Reusing the Assignment Trail in CDCL Solvers." In: *J. Satisf. Boolean Model. Comput.* 7.4 (2011), pp. 133–138. DOI: 10.3233/sat190082.

[189] Marc Thurley. "sharpSAT – Counting Models with Advanced Component Caching and Implicit BCP." In: *Theory and Applications of Satisfiability Testing – SAT 2006, 9th International Conference, Seattle, WA, USA, August 12–15, 2006, Proceedings*. Ed. by Armin Biere and Carla P. Gomes. Vol. 4121. Lecture Notes in Computer Science. Springer, 2006, pp. 424–429. DOI: 10.1007/11814948_38.

[190] Abraham Temesgen Tibebu and Görschwin Fey. "Augmenting All Solution SAT Solving for Circuits with Structural Information." In: *21st IEEE International Symposium on Design and Diagnostics of Electronic Circuits & Systems, DDECS 2018, Budapest, Hungary, April 25–27, 2018*. IEEE, 2018, pp. 117–122. DOI: 10.1109/DDECS.2018.00028.

[191] Takahisa Toda and Takehide Soh. "Implementing Efficient All Solutions SAT Solvers." In: *ACM J. Exp. Algorithmics* 21.1 (2016), 1.12:1–1.12:44. DOI: 10.1145/2975585.

[192] Grigori Tseitin. "On the complexity of derivation in propositional calculus." In: *Studies in Constructive Mathematics and Mathematical Logic* (1968), pp. 115–125.

[193] Melvin M. Vopson. "Estimation of the information contained in the visible matter of the universe." In: *AIP Advances* 11.10 (2021), p. 105317. DOI: 10.1063/5.0064475. eprint: https://doi.org/10.1063/5.0064475.

[194] Wenxi Wang, Muhammad Usman, Alyas Almaawi, Kaiyuan Wang, Kuldeep S. Meel, and Sarfraz Khurshid. "A Study of Symmetry Breaking Predicates and Model Counting." In: *Tools and Algorithms for the Construction and Analysis of Systems – 26th International Conference, TACAS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25–30, 2020, Proceedings, Part I*. Ed. by Armin Biere and David Parker. Vol. 12078. Lecture Notes in Computer Science. Springer, 2020, pp. 115–134. DOI: 10.1007/978-3-030-45190-5_7.

[195] Ingo Wegener. *The complexity of Boolean functions*. Wiley-Teubner, 1987. URL: http://ls2-www.cs.uni-dortmund.de/monographs/bluebook/.

[196] Wei Wei and Bart Selman. "A New Approach to Model Counting." In: *Theory and Applications of Satisfiability Testing, 8th International Conference, SAT 2005, St. Andrews, UK, June 19–23, 2005, Proceedings*. Ed. by Fahiem Bacchus and Toby Walsh. Vol. 3569. Lecture Notes in Computer Science. Springer, 2005, pp. 324–339. DOI: 10.1007/11499107_24.

[197] Christoph Weidenbach. "Automated Reasoning Building Blocks." In: *Correct System Design*. Vol. 9360. Lecture Notes in Computer Science. Springer, 2015, pp. 172–188.

[198] Christoph Wernhard. "The PIE Environment for First-Order-Based Proving, Interpolating and Eliminating." In: *Proceedings of the 5th Workshop on Practical Aspects of Automated Reasoning co-located with International Joint Conference on Automated Reasoning (IJCAR 2016), Coimbra, Portugal, July 2nd, 2016*. Ed. by Pascal Fontaine, Stephan Schulz, and Josef Urban. Vol. 1635. CEUR Workshop Proceedings. CEUR-WS.org, 2016, pp. 125–138. URL: http://ceur-ws.org/Vol-1635/paper-11.pdf.

[199] Jan Wielemaker, Tom Schrijvers, Markus Triska, and Torbjörn Lager. "SWI-Prolog." In: *Theory and Practice of Logic Programming (TPLP)* 12.1–2 (2012), pp. 67–96. DOI: 10.1017/S1471068411000494.

[200] Fan Xiao, Mao Luo, Chu-Min Li, Felip Manyà, and Zhipeng Lü. "MapleLRB LCM, Maple LCM, Maple LCM Dist, MapleLRB LCMoccRestart and Glucose-3.0+width in SAT Competition 2017." In: *Proceedings of SAT Competition 2017: Solver and Benchmark Descriptions*. Vol. B-2017-1. Publications Series B. Department of Computer Science, University of Helsinki, 2017, pp. 22–23. URL: https://researchportal.helsinki.fi/files/91476978/sc2017_proceedings.pdf.

[201] Jiayi Yang, Wenxi Wang, Darko Marinov, and Sarfraz Khurshid. "AlloyMC: Alloy meets model counting." In: *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8–13, 2020*. Ed. by Prem Devanbu, Myra B. Cohen, and Thomas Zimmermann. ACM, 2020, pp. 1541–1545. DOI: 10.1145/3368089.3417938.

[202] Erik Peter Zawadzki, André Platzer, and Geoffrey J. Gordon. "A Generalization of SAT and #SAT for Robust Policy Evaluation." In: *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3–9, 2013*. Ed. by Francesca Rossi. IJCAI/AAAI, 2013, pp. 2583–2590. URL: http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6838.

[203] Christoph Zengler and Wolfgang Küchlin. “Boolean Quantifier Elimination for Automotive Configuration – A Case Study.” In: *Formal Methods for Industrial Critical Systems – 18th International Workshop, FMICS 2013, Madrid, Spain, September 23–24, 2013. Proceedings*. Ed. by Charles Pecheur and Michael Dierkes. Vol. 8187. Lecture Notes in Computer Science. Springer, 2013, pp. 48–62. DOI: 10.1007/978-3-642-41010-9_4.

[204] Lintao Zhang. “Solving QBF by Combining Conjunctive and Disjunctive Normal Forms.” In: *Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, July 16–20, 2006, Boston, Massachusetts, USA*. AAAI Press, 2006, pp. 143–150. URL: http://www.aaai.org/Library/AAAI/2006/aaai06-023.php.

[205] Shengjia Zhao, Sorathan Chaturapruek, Ashish Sabharwal, and Stefano Ermon. “Closing the Gap Between Short and Long XORs for Model Counting.” In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12–17, 2016, Phoenix, Arizona, USA*. Ed. by Dale Schuurmans and Michael P. Wellman. AAAI Press, 2016, pp. 3322–3329. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12546.

INDEX

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LyX:

https://bitbucket.org/amiede/classicthesis/

Final Version as of June 2, 2022 (`classicthesis` version 4.2).