UAB
Universitat Autònoma de Barcelona

Proof Systems and Solvers for MaxSAT and MinSAT in Boolean and Many-Valued Logics

Elifnaz YANGIN

ADVERTIMENT. L'accés als continguts d'aquesta tesi queda condicionat a l'acceptació de les condicions d'ús establertes per la següent llicència Creative Commons:  https://creativecommons.org/licenses/?lang=ca

ADVERTENCIA. El acceso a los contenidos de esta tesis queda condicionado a la aceptación de las condiciones de uso establecidas por la siguiente licencia Creative Commons:  https://creativecommons.org/licenses/?lang=es

WARNING. The access to the contents of this doctoral thesis it is limited to the acceptance of the use conditions set by the following Creative Commons license:  https://creativecommons.org/licenses/?lang=en





Proof Systems and Solvers for MaxSAT and MinSAT in Boolean and Many-Valued Logics

by

Elifnaz YANGIN

A dissertation presented in partial fulfilment of the requirements for the degree of Doctor of Philosophy in Computer Science

Advisors:

Jordi COLL and Felip MANYÀ

July 2026



Programa de Doctorat en Informàtica

Abstract

This PhD thesis advances the theoretical foundations and exact solving techniques for Maximum Satisfiability (MaxSAT) and Minimum Satisfiability (MinSAT) in both Boolean and many-valued propositional logics. Although MaxSAT and MinSAT are well-established optimization frameworks, dedicated proof systems and competitive solvers for many-valued logics, such as regular and signed logics, remain significantly less developed than their Boolean counterparts. To address this gap, this thesis introduces novel resolution and tableau calculi that provide the foundation for developing exact solvers and Mixed-Integer Linear Programming (MILP) encodings for many-valued MaxSAT and MinSAT.

While the MaxSAT resolution rule has been extensively studied from a theoretical perspective, algorithmic implementations remain scarce. To overcome this limitation, we develop a highly optimized exact variable elimination solver for Boolean Weighted Partial MaxSAT and empirically demonstrate its effectiveness both as an exact solver and as a Bounded Variable Elimination (BVE) preprocessor for MaxSAT solvers. Moving beyond classical logic, we formalize the first sound and complete resolution calculus and exact variable elimination algorithm for Weighted Regular MaxSAT. We then define complete tableau calculi for Weighted Partial Regular MaxSAT and Weighted Partial Regular MinSAT. Building on these calculi, we introduce a novel MILP encoding that enables state-of-the-art MILP solvers to solve both problems, and we evaluate this approach computationally using randomly generated instances. Finally, we extend these proof systems to signed logic by establishing the first sound and complete tableau calculi for Weighted Partial Signed MaxSAT and Weighted Partial Signed MinSAT.

In conclusion, this thesis delivers both an efficient algorithmic realization of Boolean MaxSAT resolution and a substantial extension of the proof systems and exact solving techniques available for many-valued optimization, significantly narrowing the gap between Boolean and many-valued MaxSAT and MinSAT.

Resum

Aquesta tesi doctoral desenvolupa nous fonaments teòrics i tècniques de resolució exacta per als problemes de Màxima Satisfactibilitat (MaxSAT) i Mínima Satisfactibilitat (MinSAT) en lògiques proposicionals tant booleanes com multivaluades. Tot i que MaxSAT i MinSAT són marcs d'optimització ben establerts, els sistemes de prova i els *solvers* per a les lògiques multivaluades, com ara la lògica regular i la lògica signada, continuen estant considerablement menys desenvolupats que els seus homòlegs booleans. Per abordar aquesta mancança, la tesi introdueix nous càlculs de resolució i de *tableaux*, que constitueixen la base per desenvolupar tant *solvers* exactes com codificacions de Programació Lineal Entera Mixta (MILP) per als problemes MaxSAT i MinSAT multivaluats.

Tot i que la regla de resolució per a MaxSAT ha estat àmpliament estudiada des d'una perspectiva teòrica, les implementacions algorísmiques continuen sent escasses. Per superar aquesta limitació, desenvolupem un *solver* exacte d'eliminació de variables altament optimitzat per a MaxSAT Booleà Parcial Ponderat i demostrem empíricament la seva eficàcia tant com a *solver* exacte com a preprocessador per a *solvers* MaxSAT. Més enllà de la lògica clàssica, formalitzem el primer càlcul de resolució sòlid i complet, així com el primer algorisme exacte d'eliminació de variables, per a MaxSAT Regular Ponderat. A continuació, definim càlculs de *tableaux* sòlids i complets per a MaxSAT Regular Parcial Ponderat i MinSAT Regular Parcial Ponderat. A partir d'aquests càlculs, introduïm una nova codificació MILP que permet als *solvers* MILP d'última generació resoldre ambdós problemes, i n'avaluem computacionalment el rendiment mitjançant instàncies generades aleatòriament. Finalment, estenem aquests sistemes de prova a la lògica signada mitjançant el desenvolupament dels primers càlculs de *tableaux* sòlids i complets per a MaxSAT Signat Parcial Ponderat i MinSAT Signat Parcial Ponderat.

En resum, aquesta tesi proporciona tant una implementació algorísmica eficient de la resolució MaxSAT booleana com una ampliació substancial dels sistemes de prova i de les tècniques de resolució exacta disponibles per a l'optimització multivaluada, reduint de manera significativa la bretxa teòrica i algorísmica entre els problemes MaxSAT i MinSAT booleans i multivaluats.

Resumen

Esta tesis doctoral desarrolla nuevos fundamentos teóricos y técnicas de resolución exacta para los problemas de Máxima Satisfactibilidad (MaxSAT) y Mínima Satisfactibilidad (MinSAT) en lógicas proposicionales tanto booleanas como multivaluadas. Aunque MaxSAT y MinSAT son marcos de optimización bien establecidos, los sistemas de prueba y los *solvers* para las lógicas multivaluadas, tales como las lógicas regular y signada, siguen estando considerablemente menos desarrollados que sus homólogos booleanos. Para abordar esta carencia, la tesis introduce nuevos cálculos de resolución y *tableaux*, los cuales proporcionan la base para desarrollar tanto *solvers* exactos como codificaciones de Programación Lineal Entera Mixta (MILP) para los problemas MaxSAT y MinSAT multivaluados.

Si bien la regla de resolución para MaxSAT ha sido ampliamente estudiada desde una perspectiva teórica, las implementaciones algorítmicas siguen siendo escasas. Para superar esta limitación, desarrollamos un *solver* exacto de eliminación de variables altamente optimizado para MaxSAT Booleano Parcial Ponderado y demostramos empíricamente su eficacia tanto como *solver* exacto como preprocesador para *solvers* MaxSAT. Más allá de la lógica clásica, formalizamos el primer cálculo de resolución sólido y completo, así como el primer algoritmo exacto de eliminación de variables, para MaxSAT Regular Ponderado. A continuación, definimos cálculos de *tableaux* sólidos y completos para MaxSAT Regular Parcial Ponderado y MinSAT Regular Parcial Ponderado. Sobre la base de estos cálculos, introducimos una nueva codificación MILP que permite a los *solvers* MILP de última generación resolver ambos problemas, y evaluamos computacionalmente este enfoque utilizando instancias generadas aleatoriamente. Finalmente, extendemos estos sistemas de prueba a la lógica signada mediante el desarrollo de los primeros cálculos de *tableaux* sólidos y completos para MaxSAT Signado Parcial Ponderado y MinSAT Signado Parcial Ponderado.

En resumen, esta tesis proporciona tanto una implementación algorítmica eficiente de la resolución MaxSAT booleana como una ampliación sustancial de los sistemas de prueba y las técnicas de resolución exacta disponibles para la optimización multivaluada, reduciendo de manera significativa la brecha teórica y algorítmica entre los problemas MaxSAT y MinSAT booleanos y multivaluados.

Acknowledgments

Many people made this journey possible, and I carry gratitude for each of them.

When I first met my advisor, Felip Manyà, I came to him as an Erasmus student. Now I leave him as someone who has completed her PhD under his guidance. I will be grateful to him for the rest of my life for seeing things in me that I could not even see myself, and for showing me the way. He was also the one who introduced me to my other advisor, Jordi Coll. To Jordi, I owe my thanks for making me feel, over all these years, that I was being listened to without judgment.

To the wonderful people of IIIA-CSIC, who showed me that another kind of life was possible, thank you all.

I thank everyone on my committee for the time they gave me and for their thoughtful, supportive questions.

I am also grateful to Chu Min Li, Marcelo Finger, and Sandro Preto, who made my research stays abroad smooth and welcoming, and who made me feel that I was not alone in cities I knew nothing about.

To all my friends who accepted me as I am, thank you. To the friends I left behind in Turkey, to the ones I found in Spain, and to the ones I met on the other side of the ocean. But among them there is one who, I know, would do anything for me except eat red meat. Umay Arslan, thank you for teaching me to be brave, and to love myself.

To my dear family, I am grateful for all the support they have given me through these years. I know I could not have become the person I am today without my mother's guidance, my father's tenderness, and my brother's support, and I thank them with all my heart.

My most important thanks go to Mustafa Kemal Atatürk, for always reminding me that the strength I was searching for already runs in the noble blood within my veins.

Elifnaz YANGIN was supported by a grant PTA2022-022408-I funded by MICIU/AEI/10.13039/501100011033 and FSE+

Teşekkür

Bütün bunların hiçbiri pek çok insanın yardımı olmadan mümkün olamazdı. Hepsine minnettarım.

Danışmanım Felip Manyà ile tanıştığımda yanına bir Erasmus öğrencisi olarak gelmiştim. Şimdi yanından, onun rehberliğinde doktorasını tamamlamış biri olarak ayrılıyorum. Bende benim bile göremediğim şeyleri görüp bana yol gösterdiği için ona hayatımın sonuna kadar minnettar kalacağım. Diğer danışmanım Jordi Coll ile tanışmamı da kendisi sağladı. Jordi'ye, yıllar boyunca beni yargılamadan dinlediğini hissettirdiği için teşekkürü bir borç bilirim.

Bana başka bir hayatın mümkün olduğunu gösteren IIIA-CSIC'in güzel insanları, hepinize çok teşekkür ederim.

Jürimde bulunan herkese, bana ayırdıkları zaman ve düşünceli, destekleyici sorgulamaları için teşekkür ederim.

Yurt dışı araştırma ziyaretlerimin sorunsuz ve sıcak geçmesini sağlayan, hiç bilmediğim şehirlerde yalnız olmadığımı hissettiren Chu Min Li, Marcelo Finger ve Sandro Preto'ya da teşekkür ederim.

Beni olduğum gibi kabul eden bütün arkadaşlarıma teşekkür ederim. Türkiye'de geride bıraktıklarıma, İspanya'da edindiklerime ve okyanusun diğer ucunda tanıştıklarıma. Ama aralarında biri var ki, benim için yapamayacağı tek şeyin kırmızı et yemek olduğunu biliyorum. Umay Arslan, bana cesur olmayı ve kendime değer vermeyi öğrettiğin için teşekkür ederim.

Sevgili aileme, bunca yıl bana verdikleri destek için minnettarım. Annemin yol göstericiliği, babamın şefkati ve kardeşimin desteği olmadan şu anki olduğum kişi olamayacağımı biliyorum ve onlara tüm kalbimle teşekkür ediyorum.

En önemli teşekkürüm Ata'ma, aradığım kudretin damarlarımda akan asil kanda mevcut olduğunu bana daima hatırlattığı için.

Elifnaz YANGIN, MICIU/AEI/10.13039/501100011033 ve FSE+ tarafından finanse edilen PTA2022-022408-I numaralı burs ile desteklenmiştir.

Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Objectives . . . 3
- 1.3 Contributions . . . 4
- 1.4 Publications . . . 4
- 1.5 Outline of the Thesis . . . 5

2 Preliminaries **7**
- 2.1 Boolean Propositional Logic . . . 7
- 2.2 Signed Propositional Logic . . . 10
- 2.3 Regular Propositional Logic . . . 11
- 2.4 Resolution for Boolean Propositional Logic . . . 12
- 2.5 Tableaux for Boolean Propositional Logic . . . 13

3 A Resolution Solver for Boolean MaxSAT **17**
- 3.1 Introduction . . . 17
- 3.2 SAT Variable Elimination . . . 18
- 3.3 MaxSAT Resolution Rule . . . 20
  - 3.3.1 Weighted Partial MaxSAT Resolution . . . 22
- 3.4 MSElim: a Variable Elimination Solver for MaxSAT . . . 23
  - 3.4.1 A Generic Variable Elimination Algorithm . . . 23
  - 3.4.2 Formula Representation . . . 26
  - 3.4.3 Variable Selection Heuristics . . . 29
  - 3.4.4 Variable Saturation . . . 31
  - 3.4.5 Post-Saturation Process . . . 38
- 3.5 Experimental Evaluation . . . 41
  - 3.5.1 MSElim as a Solver . . . 41
  - 3.5.2 MSElim as a BVE Preprocessor . . . 45
- 3.6 Conclusions . . . 46

4 Resolution for Regular MaxSAT **48**
- 4.1 Introduction . . . 48
- 4.2 Regular MaxSAT Resolution . . . 49
- 4.3 A Variable Elimination Algorithm for Regular MaxSAT . . . 56

4.4 Weighted Regular MaxSAT Resolution 58
4.5 Conclusions 59

5 Tableaux for Regular MaxSAT and MinSAT 60
5.1 Introduction 60
5.2 A Regular MaxSAT tableau calculus 61
5.3 A Weighted Partial Regular MaxSAT Tableau Calculus 64
5.3.1 Soundness & completeness 67
5.4 A Weighted Partial Regular MinSAT Tableau Calculus 70
5.4.1 Soundness and completeness 73
5.5 Clausal Tableau Calculi 77
5.6 Solving Regular MaxSAT and Regular MinSAT with MILP 79
5.6.1 An MILP Encoding for Regular MaxSAT 80
5.6.2 An MILP encoding for Weighted Partial Regular MaxSAT . . 86
5.6.3 An MILP Encoding for Weighted Partial Regular MinSAT . . 88
5.6.4 Experimental Evaluation 88
5.7 Conclusions 91

6 Tableaux for Signed MaxSAT and MinSAT 92
6.1 Introduction 92
6.2 A Signed MaxSAT Tableau Calculus 93
6.3 A Weighted Partial Signed MaxSAT Tableau Calculus 95
6.3.1 Soundness and Completeness 97
6.4 A Signed MinSAT Tableau Calculus 102
6.5 A Weighted Partial Signed MinSAT Tableau Calculus 103
6.6 Conclusions 110

7 Conclusions and Future Work 111
7.1 Conclusions 111
7.2 Future Work 112

Alphabetical Index 122

List of Figures

2.1 Truth tables for $N = \{0, \frac{1}{2}, 1\}$. 10
2.2 Completed SAT tableau for $\{x_1, x_2, \neg x_1 \wedge \neg x_2\}$. 14
2.3 MaxSAT tableau construction for $\{x_1, x_2, \neg x_1 \wedge \neg x_2\}$. 16

3.1 Bipartite graph of heuristic 1 30
3.2 Variable interaction graph of heuristic 3 30
3.3 Neighbourhood abstraction for heuristic 2 31
3.4 Post-saturation process for eliminating a variable in MSElim. 40
3.5 Ablation study of MSElim for unweighted instances 43
3.6 Ablation study of MSElim for unweighted instances 44
3.7 Results on MSElim as a BVE preprocessor. 47

4.1 Rule 1 of Resolution Regular MaxSAT in clausal form 51
4.2 Examples of rule applications in Resolution Regular MaxSAT 51
4.3 Example of a saturated multiset of regular clauses 54
4.4 Proof system for Weighted Regular MaxSAT. 58

5.1 Tableau expansion rules for Regular MaxSAT 62
5.2 A completed tableau for Regular MaxSAT 64
5.3 Derivation of the $\wedge$-rule from the $\neg$- and $\vee$-rules. 65
5.4 Tableau expansion rules for Weighted Partial Regular MaxSAT. 66
5.5 Tableau expansion rules for Weighted Partial Regular MinSAT. 71
5.6 A completed tableau for Weighted Partial Regular MinSAT 74
5.7 Tableau expansion rules for Clausal Regular MaxSAT. 78
5.8 A tableau calculus for Clausal Weighted Partial Regular MaxSAT. . . 79
5.9 A tableau calculus for Clausal Weighted Partial Regular MinSAT. . . 80
5.10 Regular Max-2SAT: Satisfaction rate against the clause-to-variable ratio 89
5.11 Regular Max-3SAT: Satisfaction rate against the clause-to-variable ratio 90
5.12 Regular Max-2SAT: Solving time against the clause-to-variable ratio . 90
5.13 Regular Max-3SAT: Solving time against the clause-to-variable ratio 91

6.1 A complete tableau calculus for Signed MaxSAT. 94
6.2 A completed Signed MaxSAT tableau 95
6.3 A complete tableau calculus for Weighted Partial Signed MaxSAT. . . 96

6.4 A completed Weighted Partial Signed MaxSAT tableau 101
6.5 A complete tableau calculus for Signed MinSAT. 103
6.6 A completed Signed MinSAT tableau 104
6.7 A complete tableau calculus for Weighted Partial Signed MinSAT. . . 105
6.8 A completed Weighted Partial Signed MinSAT tableau 106

List of Tables

2.1 α-formulas and β-formulas. 14
2.2 Tableau expansion rules for SAT. 14
2.3 Tableau expansion rules for MaxSAT. 15

3.1 MSElim's literal encoding 27
3.2 MSElim's clause representation 27
3.3 Results on MSElim as a BVE preprocessor 45

List of Algorithms

3.1 An exact variable elimination algorithm for SAT 19
3.2 An exact variable elimination algorithm for Weighted Partial MaxSAT . 24
3.3 A generic algorithm for Weighted Partial MaxSAT saturation 32

4.1 An exact variable elimination algorithm for Regular MaxSAT 57

Chapter 1

Introduction

This chapter lays the foundation for the research presented in this thesis. It begins by presenting the underlying motivation and defining both the main and specific objectives. It then details the key theoretical and practical contributions, together with the peer-reviewed publications derived from this work. Finally, the chapter concludes with an outline of the thesis structure and provides a brief description of the contents of the upcoming chapters.

1.1 Motivation

Logic has played a central role in the development of Artificial Intelligence (AI), providing the foundations of symbolic knowledge representation and automated reasoning. Although a wide range of expressive non-classical logics have long been investigated for knowledge representation and reasoning, the remarkable advances in SAT solving since the late 20th century have demonstrated that many large and challenging AI problems can be effectively tackled using Boolean propositional logic [17]. This progress led to a paradigm shift, greatly expanding the practical impact of automated reasoning and establishing SAT solving as one of the fundamental technologies of modern AI, with its core techniques now routinely taught in university-level AI courses.

The Satisfiability problem (SAT) asks whether there exists a truth assignment that satisfies a given set of propositional formulas, typically in clausal form. SAT is one of the most extensively studied problems in Computer Science, being central from both theoretical and practical perspectives. Many decision problems can be encoded into Boolean propositional logic and solved using highly optimized SAT solvers. As a result, SAT has become a powerful generic method for solving decision problems in fields such as automatic circuit design [26, 83, 100], formal verification of hardware and software [19, 16, 28, 29, 48, 56, 93, 94], scheduling [12, 47, 97], planning [57, 58, 86], bioinformatics [76, 77], cryptography [42], manufacturing [41], and mathematics [52, 53, 59, 96].

Today, driven by powerful algorithmic advances such as Conflict-Driven Clause

Learning (CDCL), sophisticated heuristics, and highly optimized data structures, modern SAT solvers can regularly handle instances containing millions of variables and clauses within practical running times. Rather than relying on brute-force exploration, these systems dynamically learn from logical contradictions encountered during the search, allowing them to prune massive regions of the search space. Consequently, SAT has evolved into a highly scalable technology for solving many practically relevant problems. For many complex classes of combinatorial problems, these solvers now routinely outperform other domain-specific, state-of-the-art problem-solving methods.

While SAT is well-suited for decision problems, many real-world applications require optimizing an objective under logical constraints. This need motivates optimization extensions of SAT, among which the Maximum Satisfiability problem (MaxSAT) has become the most prominent. MaxSAT aims to find an assignment that maximizes the number of satisfied formulas in a given multiset of propositional formulas, or equivalently minimizes the number of unsatisfied formulas. In the weighted partial setting, the formulas are partitioned into hard and soft formulas, where each soft formula is assigned a weight corresponding to the penalty incurred when it is violated. An optimal solution is then an assignment that satisfies all hard formulas and minimizes the sum of weights of the unsatisfied soft formulas.

Over the last two decades, MaxSAT has gained popularity as a generic problem-solving technology for combinatorial optimization [10, 64]. When equipped with competitive solvers (e.g., [9, 13, 30, 62, 63, 82, 85]), MaxSAT provides a scalable and efficient framework for solving a broad spectrum of combinatorial optimization problems, via their reduction to MaxSAT, in areas such as bioinformatics [80], combinatorial testing [6], planning [98], scheduling [20] and team formation [78].

Another variant of SAT is the Minimum Satisfiability problem (MinSAT) [73]. In its unweighted form, the goal of MinSAT is to find an assignment that minimizes the number of satisfied formulas in the input multiset, or equivalently maximizes the number of unsatisfied formulas. Similarly to MaxSAT, MinSAT can be extended to its partial and weighted variants. In this setting, an optimal solution is an assignment that satisfies all hard formulas while minimizing the sum of weights of the satisfied soft formulas.

At first glance, it may appear that the encodings and solving techniques developed for MaxSAT could also be applied to MinSAT. If so, a dedicated investigation of MinSAT would seem unnecessary. Nevertheless, most research to date indicates that the techniques required for each problem can be substantially different or complementary. For example, the performance profiles of MaxSAT and MinSAT vary significantly across several optimization problems modeled under these formalisms [8, 55, 74]. At the proof-theoretic level, the proof systems for MaxSAT and MinSAT also exhibit fundamental differences [65, 66].

While the aforementioned Boolean formalisms have proven highly successful, classical two-valued logic does not always provide the most suitable knowledge representation framework. A natural extension is offered by many-valued logics, which accommodate arbitrary finite or infinite cardinalities within the truth value set. Furthermore, they feature distinct interpretation functions for each connective that depend entirely on the underlying logic, giving rise to richer expressive power. In this context, we focus on two prominent many-valued formalisms: signed logic and regular logic.

Signed logic generalizes Boolean logic by considering a finite set N of truth values and associating each formula with a sign $S \subseteq N$. Within this framework, an assignment is a mapping that assigns a truth value from N to each propositional variable, which is extended to arbitrary formulas by interpreting conjunction, disjunction, and negation through the minimum, maximum, and Łukasiewicz negation functions, respectively. An assignment v satisfies a signed formula $S{:}A$ if and only if $v(A) \in S$. Given a multiset of signed formulas, the MaxSAT (MinSAT) problem aims to find an assignment that maximizes (minimizes) the number of satisfied formulas.

Regular logic introduces a total order over the truth value set N, which may be finite or infinite, and restricts signs to subsets of the form $\{j \in N \mid j \geq i\}$ or $\{j \in N \mid j \leq i\}$ for some $i \in N$, known as regular signs. While regular logic can be viewed as a specialization of signed logic with respect to its sign structure, it broadens the scope by explicitly accommodating infinite truth value sets. In particular, restricting formulas to regular signs enables significant simplifications, allowing for the definition of more efficient proof systems.

Despite the greater expressive power of regular and signed logics, the proof systems and exact solving technologies available for optimization in these formalisms remain far less developed than those for Boolean logic. Bridging this gap is one of the primary goals of this thesis. To this end, we pursue two complementary directions. First, we extend the proof-theoretic foundations of MaxSAT and MinSAT to regular and signed logics by introducing new resolution and tableau calculi. Second, we develop Mixed Integer Linear Programming (MILP) encodings for these problems, enabling state-of-the-art MILP solvers to compute optimal solutions.

1.2 Objectives

The main objective of this thesis is to advance the proof-theoretic foundations and exact solving techniques for Boolean and many-valued MaxSAT and MinSAT. This objective is pursued through the formalization of novel sound and complete resolution and tableau calculi, together with the development of exact solvers that demonstrate the practical applicability of these theoretical advances. More specifically, the thesis addresses the following objectives:

- To design and implement an exact variable-elimination algorithm for Boolean Weighted Partial MaxSAT based on the MaxSAT resolution rule [25], and to empirically evaluate its performance on benchmark instances from the MaxSAT Evaluation.
- To establish the first sound and complete resolution calculus and an exact variable-elimination framework for Weighted Regular MaxSAT.
- To develop complete tableau calculi for Weighted Partial Regular MaxSAT and Weighted Partial Regular MinSAT, and to derive Mixed-Integer Linear Programming (MILP) encodings from these calculi, enabling state-of-the-art MILP solvers to solve both problem variants.

- To develop complete tableau calculi for Weighted Partial Signed MaxSAT and Weighted Partial Signed MinSAT.

1.3 Contributions

The main contributions of this thesis advance the state of the art in MaxSAT and MinSAT solving and can be summarized as follows:

- The design, development and empirical evaluation of a highly optimized implementation of an exact variable elimination solver, named MSElim, for Boolean MaxSAT.
- The design, development and empirical evaluation of a Bounded Variable Elimination (BVE) preprocessor for Boolean MaxSAT solvers.
- The formalization of the first sound and complete resolution calculus for Weighted Regular MaxSAT, complemented by the introduction of the first exact variable elimination framework for this problem.
- The definition of the first sound and complete tableau calculi for both Weighted Partial Regular MaxSAT and Weighted Partial Regular MinSAT. Based on the principles of these calculi, a novel MILP encoding is introduced to solve both problem variants using state-of-the-art MILP solvers.
- The extension of these theoretical frameworks to signed logic through the definition of the first sound and complete tableau calculi for Weighted Partial Signed MaxSAT and Weighted Partial Signed MinSAT.

1.4 Publications

The research and contributions presented in this thesis have led to several peer-reviewed publications in international journals [32, 35] and conference proceedings [31, 33, 34, 36]. These works are divided below into journal articles, conference papers, and manuscripts currently under development.

Journal articles

- Jordi Coll, Chu Min Li, Felip Manyà, and Elifnaz Yangin. Complete tableau calculi for Regular MaxSAT and Regular MinSAT. *Cognitive Systems Research*, 90:101319, 2025.
- Jordi Coll, Chu Min Li, Felip Manyà, and Elifnaz Yangin. MaxSAT resolution for regular propositional logic. *International Journal of Approximate Reasoning*, 162:109010, 2023.

Conference papers

- Jordi Coll, Chu Min Li, Felip Manyà, and Elifnaz Yangin. A Complete Tableau Calculus for Signed MaxSAT. In *Proceedings of the 55th IEEE International Symposium on Multiple-Valued Logic (ISMVL 2025)*, pages 38–43, 2025.
- Jordi Coll, Chu Min Li, Felip Manyà, and Elifnaz Yangin. A Tableau Calculus for Non-Clausal Regular MaxSAT. In *Proceedings of the 54th IEEE International Symposium on Multiple-Valued Logic (ISMVL 2024)*, pages 137–142, 2024.
- Jordi Coll, Chu Min Li, Felip Manyà, and Elifnaz Yangin. A Complete Tableau Calculus for Non-Clausal Regular MinSAT. In *Proceedings of the 26th International Conference of the Catalan Association for Artificial Intelligence (CCIA 2024)*, pages 278–287, 2024.
- Jordi Coll, Chu Min Li, Felip Manyà, and Elifnaz Yangin. A Complete Tableau Calculus for the Regular MaxSAT Problem. In *Proceedings of the 25th International Conference of the Catalan Association for Artificial Intelligence (CCIA 2023)*, pages 359–368, 2023.

Manuscripts in preparation

Beyond the peer-reviewed publications listed above, the research reported in this thesis has led to two additional manuscripts currently in preparation. The first presents the design, implementation, and empirical evaluation of the variable elimination solver for Boolean MaxSAT, while the second focuses on the development and performance analysis of the MILP-based approach for Regular MaxSAT.

1.5 Outline of the Thesis

This section describes the structure of the document and provides a brief description of the contents of the upcoming chapters.

Chapter 2: Preliminaries. This chapter presents the fundamental logical background required throughout this thesis. It begins with an overview of standard Boolean propositional logic, which serves as the foundation for subsequent formalisms. It then introduces both signed and regular propositional logics. Finally, the chapter describes the resolution and semantic tableau calculi for Boolean propositional logic, providing the basis for the proof systems developed in the remainder of the thesis.

Chapter 3: Resolution Solver for Boolean MaxSAT. This chapter introduces the MaxSAT resolution rule and its extension to Weighted Partial MaxSAT. It then describes a variable elimination algorithm based on this resolution rule and presents MSElim, an efficient implementation of the algorithm that incorporates several features which can be turned on or off to assess their individual impact on performance.

Finally, the chapter provides a comprehensive empirical evaluation of MSElim, both as a complete solver and as a Bounded Variable Elimination (BVE) preprocessor for state-of-the-art MaxSAT solvers.

Chapter 4 Resolution Calculus for Regular MaxSAT. This chapter introduces a novel MaxSAT resolution calculus for regular propositional logic and establishes its soundness and completeness. It also describes a variable elimination algorithm that implements the rules of the defined calculus and extends the calculus to Weighted Regular MaxSAT.

Chapter 5 Tableau Calculi for Regular MaxSAT and Regular MinSAT. This chapter introduces novel tableau calculi for solving the MaxSAT and MinSAT problems within the framework of regular propositional logic, establishing their soundness and completeness for both Weighted Partial Regular MaxSAT and Weighted Partial Regular MinSAT. Additionally, it proposes new Mixed-Integer Linear Programming (MILP) encodings that enable state-of-the-art MILP solvers to efficiently handle these weighted partial optimization problems. Finally, it reports an empirical evaluation of the proposed MILP-based approach on randomly generated Max-2SAT and Max-3SAT instances.

Chapter 6 Tableau Calculi for Signed MaxSAT and Signed MinSAT. This chapter introduces novel tableau calculi for solving the MaxSAT and MinSAT problems within the framework of signed propositional logic, establishing their soundness and completeness for both Weighted Partial Signed MaxSAT and Weighted Partial Signed MinSAT. Since their expansion rules depend on the cardinality of the truth value set and lack uniformity, these calculi operate over a fixed three-valued domain.

Chapter 7 Conclusions and Future Work. This chapter summarizes the main contributions achieved throughout this thesis, contextualizing them within the scope of the original research objectives. Furthermore, it outlines several promising research directions and open problems, highlighting potential avenues for future work.

Chapter 2

Preliminaries

This chapter presents the fundamental logical background required throughout this thesis. It begins with an overview of standard Boolean propositional logic, which serves as the foundation for subsequent formalisms. It then introduces both signed and regular propositional logics. Finally, the chapter describes the resolution and semantic tableau calculi for Boolean propositional logic, providing the basis for the proof systems developed in the remainder of the thesis.

2.1 Boolean Propositional Logic

This section presents the syntax and semantics of Boolean propositional logic and formally defines the Satisfiability (SAT) problem, along with its optimization variants: Maximum Satisfiability (MaxSAT) and Minimum Satisfiability (MinSAT).

Definition 2.1.1. *Let $\mathscr{V} = \{x_i\}_{i\in\mathbb{N}}$ be a countable set of propositional variables, and let $\neg, \vee, \wedge, \rightarrow$ be the negation, disjunction, conjunction, and implication operators, respectively. The language $\mathscr{L}$ of propositional formulas is inductively defined as follows:*

- *Every variable $x_i \in \mathscr{V}$ is a formula.*
- *If ϕ is a formula, then $\neg\phi$ is a formula.*
- *If ϕ and ψ are formulas, then $(\phi \vee \psi)$, $(\phi \wedge \psi)$, and $(\phi \rightarrow \psi)$ are formulas.*

Definition 2.1.2. *The* Boolean propositional logic *consists of the language $\mathscr{L}$ along with a semantics determined by an assignment $v : \mathscr{V} \rightarrow \{0,1\}$, which is inductively*

extended to all propositional formulas as follows:

$$\begin{aligned} v(\neg\phi) &= 1 - v(\phi); \\ v(\phi \vee \psi) &= \max\{v(\phi), v(\psi)\}; \\ v(\phi \wedge \psi) &= \min\{v(\phi), v(\psi)\}; \\ v(\phi \rightarrow \psi) &= \max\{1 - v(\phi), v(\psi)\}. \end{aligned}$$

Definition 2.1.3. *Given a set Φ of propositional formulas, the* Satisfiability *problem, or* SAT*, is to determine whether there exists an assignment to the variables occurring in Φ that satisfies Φ. If such an assignment exists, Φ is said to be* satisfiable*; otherwise, Φ is said to be* unsatisfiable*.*

Definition 2.1.4. *Given a formula ϕ, an assignment that satisfies ϕ is called a model of ϕ. A formula is a tautology if it is satisfied by every assignment.*

Definition 2.1.5. *Given a multiset Φ of propositional formulas, the* Maximum Satisfiability *problem, or* MaxSAT*, is to find an assignment to the variables occurring in Φ that maximizes the number of satisfied formulas in Φ (or equivalently, minimizes the number of unsatisfied formulas). Such an assignment is called an* optimal MaxSAT assignment *for Φ.*

Definition 2.1.6. *Given a multiset Φ of propositional formulas, the* Minimum Satisfiability *problem, or* MinSAT*, is to find an assignment to the variables occurring in Φ that minimizes the number of satisfied formulas in Φ (or equivalently, maximizes the number of unsatisfied formulas). Such an assignment is called an* optimal MinSAT assignment *for Φ.*

While the SAT problem only requires determining whether a formula can be satisfied, MaxSAT and MinSAT are optimization problems that count the number of satisfied or violated formulas, thereby requiring multisets to correctly preserve and account for duplicate formulas.

Definition 2.1.7. *A* weighted formula *is a pair (ϕ, w), where ϕ is a propositional formula and $w \in \mathbb{N}^+$ is its weight. in MaxSAT, this weight corresponds to the penalty incurred when the formula is not satisfied, whereas in MinSAT it corresponds to the weight incurred upon satisfying the formula.*

Definition 2.1.8. *Given a multiset Φ of weighted formulas,* Weighted MaxSAT *is to find an assignment to the variables occurring in Φ that minimizes the sum of weights of the unsatisfied formulas. Such an assignment is called a* Weighted MaxSAT optimal assignment *for Φ.*

Definition 2.1.9. *Given a multiset Φ of weighted formulas,* Weighted MinSAT *is to find an assignment to the variables occurring in Φ that minimizes the sum of weights of the satisfied formulas. Such an assignment is called a* Weighted MinSAT optimal assignment *for Φ.*

When a multiset contains both *hard* and *soft* formulas, the problem is termed *Weighted Partial MaxSAT/MinSAT*. A hard formula is represented as a pair (ϕ, ∞), where the infinity symbol indicates that ϕ must be satisfied. In contrast, a soft formula is a weighted formula (ϕ, w) with a weight $w \in \mathbb{N}^+$, representing the penalty for its violation.

Definition 2.1.10. *Given a multiset Φ of hard and soft formulas,* Weighted Partial MaxSAT *is to find an assignment to the variables occurring in Φ that satisfies all the hard formulas and minimizes the sum of weights of the unsatisfied soft formulas. Such an assignment is called a* Weighted Partial MaxSAT optimal assignment *for Φ.*

Definition 2.1.11. *Given a multiset Φ of hard and soft formulas,* Weighted Partial MinSAT *is to find an assignment to the variables occurring in Φ that satisfies all the hard formulas and minimizes the sum of weights of the satisfied soft formulas. Such an assignment is called a* Weighted Partial MinSAT optimal assignment *for Φ.*

The SAT, MaxSAT, and MinSAT problems are frequently defined for clausal forms, which constitute a restriction of propositional logic defined through the following concepts:

Definition 2.1.12. *A literal l is a propositional variable x_i or its negation $\neg x_i$. A clause c is a disjunction (or a set) of literals. A formula is in Conjunctive Normal Form (CNF) when it is represented as a conjunction (or a set) of clauses.*

Note that clauses are presented both as a disjunction of literals and as a set of literals. We occasionally mix these two representations.

Definition 2.1.13. *The total number of literals in a clause c is its size, denoted by $|c|$. A clause of size one is called a unit clause. Similarly, a clause of size two is called binary, and a clause of size three is called ternary. The size of a CNF formula Φ, denoted by $|\Phi|$, is defined as the total number of literal occurrences in the formula, which is equal to the sum of the sizes of all its clauses.*

Definition 2.1.14. *Let v be an assignment. A literal $l_i = x_i$ is satisfied by v if and only if $v(x_i) = 1$, and a literal $l_i = \neg x_i$ is satisfied if and only if $v(x_i) = 0$. A clause is satisfied by v if v satisfies at least one of its literals, and v satisfies a CNF formula if it satisfies every clause in it. If an assignment does not satisfy a formula ϕ, we say that ϕ is falsified. The empty clause, denoted by $\square$, is a clause containing no literals and is falsified by every assignment. Two CNF formulas are equivalent if they are satisfied by the exact same set of assignments.*

Definition 2.1.15. *An assignment is complete when every variable that occurs in formula ϕ is assigned a truth value; otherwise, the assignment is partial. Under a partial assignment, the clauses of a CNF formula can be satisfied, falsified, or unresolved. A clause is unresolved if it contains no satisfied literals and at least one of its literals corresponds to an unassigned variable. These unassigned literals are known as its free literals. Under a partial assignment, an unresolved clause becomes unit, binary, or ternary if the number of its free literals is one, two, or three, respectively.*

Figure 2.1: Truth tables for $N = \{0, \frac{1}{2}, 1\}$.

$\wedge$	0	$\frac{1}{2}$	1
0	0	0	0
$\frac{1}{2}$	0	$\frac{1}{2}$	$\frac{1}{2}$
1	0	$\frac{1}{2}$	1

$\vee$	0	$\frac{1}{2}$	1
0	0	$\frac{1}{2}$	1
$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	1
1	1	1	1

	$\neg$
0	1
$\frac{1}{2}$	$\frac{1}{2}$
1	0

2.2 Signed Propositional Logic

Signed logic is a many-valued logic that considers a finite set N of truth values, where every propositional formula ϕ is annotated with a non-empty subset of truth values $S \subseteq N$. A many-valued assignment v satisfies a signed formula $S : \phi$ if and only if $v(\phi) \in S$. Syntactically, propositional formulas are constructed following the standard rules outlined in Definition 2.1.1.

The remainder of this section presents the syntax and semantics of signed propositional logic, defining the signed Satisfiability problem (Signed SAT) along with its optimization variants: Signed Maximum Satisfiability (Signed MaxSAT) and Signed Minimum Satisfiability (Signed MinSAT).

Definition 2.2.1. *Given a finite set N of truth values, a* sign *is a non-empty subset $S \subseteq N$. A* signed formula *is an expression of the form $S : \phi$, where S is a sign and ϕ is a propositional formula. When ϕ is a variable, the resulting signed formula is a signed literal.*

Throughout this work, unless otherwise stated, whenever a finite set of truth values of cardinality n is considered, we fix $N = \{0, \frac{1}{n-1}, \ldots, \frac{n-1}{n-1}\}$ as the set of equidistant rational numbers in the unit interval $[0, 1]$.

Definition 2.2.2. *The* n-valued signed propositional logic *extends the language $\mathscr{L}$ from Definition 2.1.1 by annotating each formula with a sign. Its semantics is determined by a many-valued assignment $v : \mathscr{V} \to N$, which is inductively extended to all propositional formulas as follows:*

$$
\begin{aligned}
v(\neg\phi) &= 1 - v(\phi);\\
v(\phi \vee \psi) &= \max\{v(\phi), v(\psi)\};\\
v(\phi \wedge \psi) &= \min\{v(\phi), v(\psi)\};\\
v(\phi \to \psi) &= \max\{1 - v(\phi), v(\psi)\}.
\end{aligned}
$$

An assignment v satisfies signed formula $S : \phi$ if and only if $v(\phi) \in S$. A signed formula is satisfiable if there exists an assignment that satisfies it; otherwise, it is unsatisfiable.

Alternatively, the interpretation of the connectives can be defined using truth tables. Figure 2.1 displays the truth tables for $N = \{0, \frac{1}{2}, 1\}$.

Example 2.2.3. *Let* $N = \{0, \frac{1}{2}, 1\}$ *and let* $\Phi = \big\{\{0,1\}{:}x_1 \vee x_2,\ \{\frac{1}{2}\}{:}\neg x_1 \vee x_3,\ \{0, \frac{1}{2}\}{:}x_3\big\}$ *be a set of signed formulas. The assignment* $v(x_1) = 1$, $v(x_2) = 1$, $v(x_3) = \frac{1}{2}$ *satisfies* Φ. *Adding the signed formula* $\{0\}{:}\neg x_3$ *to* Φ *makes it unsatisfiable.*

Definition 2.2.4. *Given a set* Φ *of signed formulas, the* Signed Satisfiability *problem, or* Signed SAT, *is to determine whether there exists an assignment to the variables occurring in* Φ *that satisfies* Φ.

Definition 2.2.5. *Given a multiset* Φ *of signed formulas, Signed MaxSAT is to find an assignment that minimizes the number of unsatisfied formulas in* Φ.

Definition 2.2.6. *Given a multiset* Φ *of signed formulas, Signed MinSAT is to find an assignment that minimizes the number of satisfied formulas in* Φ.

Definition 2.2.7. *A* weighted signed formula *is a pair* $(S{:}\phi, w)$, *where* $S{:}\phi$ *is a signed propositional formula and* $w \in \mathbb{N}^+$ *is its weight, which corresponds to the penalty to be paid if the formula is not satisfied. Conversely, a hard formula is a pair* $(S{:}\phi, \infty)$ *and cannot be falsified by any optimal assignment.*

Definition 2.2.8. *Given a multiset* Φ *of hard and soft signed formulas,* Weighted Partial Signed MaxSAT *is to find an assignment to the variables occurring in* Φ *that satisfies all the hard formulas and minimizes the sum of weights of the unsatisfied soft formulas. Such an assignment is called a* Weighted Partial Signed MaxSAT optimal assignment *for* Φ.

Definition 2.2.9. *Given a multiset* Φ *of hard and soft signed formulas,* Weighted Partial Signed MinSAT *is to find an assignment to the variables occurring in* Φ *that satisfies all the hard formulas and minimizes the sum of weights of the satisfied soft formulas. Such an assignment is called a* Weighted Partial Signed MinSAT optimal assignment *for* Φ.

2.3 Regular Propositional Logic

Regular logic introduces a total order over the set N of truth values, which can now be either finite or infinite, and restricts the signs to subsets of the form $\{j \in N \mid j \geq i\}$ or $\{j \in N \mid j \leq i\}$ for some $i \in N$, known as regular signs. While regular logic can be viewed as a specialization of signed logic regarding its sign structure, it broadens the scope by explicitly accommodating infinite truth value sets.

The remainder of this section presents the syntax and semantics of regular propositional logic, defining the Regular Satisfiability problem (Regular SAT) along with its optimization variants: Regular Maximum Satisfiability (Regular MaxSAT) and Regular Minimum Satisfiability (Regular MinSAT).

When the set of truth values is finite, we fix $N = \{0, \frac{1}{n-1}, \ldots, \frac{n-1}{n-1}\}$ as the set of equidistant rational numbers in the unit interval $[0,1]$; otherwise, when it is infinite, we set $N = [0,1]$.

Definition 2.3.1. *Given a set N of truth values equipped with the natural total order $\leq$, a regular sign is a subset of N either of the form $\{j \in N \mid j \geq i\}$, denoted by $\geq i$, or $\{j \in N \mid j \leq i\}$, denoted by $\leq i$, for some $i \in N$. Given a propositional formula ϕ, a regular formula is an expression of the form $\geq i : \phi$ or $\leq i : \phi$. Signs of the form $\geq i$ are said to have* positive polarity*, while those of the form $\leq i$ have* negative polarity.

Definition 2.3.2. *An assignment v satisfies a regular formula of the form $\geq i : \phi$ if and only if $v(\phi) \geq i$, and v satisfies a regular formula of the form $\leq i : \phi$ if and only if $v(\phi) \leq i$. A regular formula is satisfiable if there exists an assignment that satisfies it; otherwise, it is unsatisfiable.*

Example 2.3.3. *Let $N = [0,1]$ and let $\Phi = \{\geq 0.2 : x_1 \wedge (x_2 \vee \neg x_3),\ \leq 0.1 : x_1,\ \geq 0.5 : \neg x_2 \vee x_3\}$ be a multiset of regular formulas. The assignment v defined by $v(x_1) = v(x_2) = v(x_3) = 0$ falsifies the first formula and satisfies the remaining two. The assignment v such that $v(x_1) = v(x_2) = v(x_3) = 0.5$ falsifies the second formula while satisfying the other two.*

Definition 2.3.4. *Given a set Φ of regular formulas, the* Regular Satisfiability *problem, or* Regular SAT*, is to determine whether there exists an assignment to the variables occurring in Φ that satisfies Φ.*

Definition 2.3.5. *Given a multiset Φ of regular formulas, Regular MaxSAT is to find an assignment that minimizes the number of unsatisfied formulas in Φ.*

Definition 2.3.6. *Given a multiset Φ of regular formulas, Regular MinSAT is to find an assignment that minimizes the number of satisfied formulas in Φ.*

Definition 2.3.7. *A* weighted regular formula *is a pair $(S : \phi, w)$, where $S : \phi$ is a regular propositional formula and $w \in \mathbb{N}^+$ is its weight, which corresponds to the penalty to be paid if the formula is not satisfied. Conversely, a hard formula is a pair $(S : \phi, \infty)$ and cannot be falsified by any optimal assignment.*

Definition 2.3.8. *Given a multiset Φ of hard and soft regular formulas Φ,* Weighted Partial Regular MaxSAT *is to find an assignment to the variables occurring in Φ that satisfies all the hard formulas and minimizes the sum of weights of the unsatisfied soft formulas. Such an assignment is called a* Weighted Partial Regular MaxSAT optimal assignment *for Φ.*

Definition 2.3.9. *Given a multiset Φ of hard and soft regular formulas,* Weighted Partial Regular MinSAT *is to find an assignment to the variables occurring in Φ that satisfies all the hard formulas and minimizes the sum of weights of the satisfied soft formulas. Such an assignment is called a* Weighted Partial Regular MinSAT optimal assignment *for Φ.*

2.4 Resolution for Boolean Propositional Logic

The resolution calculus is a refutation system based on a single inference rule, called resolution [87]. It operates exclusively on formulas in Conjunctive Normal Form

(CNF). In this setting, a *literal* l is either a propositional variable x or its negation $\neg x$. A *clause* c is a disjunction of literals, typically represented as the set of its literals and a *CNF formula* is a set of clauses. The empty clause, denoted by $\square$, represents an inherent contradiction and is unsatisfiable.

Definition 2.4.1. *Given two clauses c_1, c_2, called* parent clauses*, then r is a* resolvent *of c_1 and c_2 if there is one literal $\ell \in c_1$ such that $\neg\ell \in c_2$, and r has the form*

$$r = (c_1 \setminus \{\ell\}) \cup (c_2 \setminus \{\neg\ell\}).$$

The application of the resolution rule to the clauses $x \vee c_1$ and $\neg x \vee c_2$ is usually represented as follows:

$$\frac{x \vee c_1 \quad \neg x \vee c_2}{c_1 \vee c_2}$$

A proof by resolution, or refutation, of a CNF formula ϕ is a sequence of rule applications that starts with the clauses in ϕ and ends with the derivation of the empty clause.

Definition 2.4.2. *A* proof by resolution*, or* refutation*, from a set C of initial clauses is a finite sequence of clauses $c_1, c_2, \ldots, c_n$ such that each clause c_i is either an element of C or a resolvent derived from two preceding clauses c_j and c_k, where $j,k < i$, via the resolution rule, and the final clause c_n is the empty clause.*

The resolution calculus is both sound and complete: a formula ϕ is unsatisfiable if and only if the empty clause can be derived from it.

Example 2.4.3. *Given the set of clauses $C = \{\neg x_1, \neg x_2, x_1 \vee x_2\}$, we prove that C is unsatisfiable by constructing a resolution refutation as follows:*

1. *From the input clauses $x_1 \vee x_2$ and $\neg x_1$, resolving on x_1 yields the resolvent x_2.*
2. *From the newly derived clause x_2 and the input clause $\neg x_2$, resolving on x_2 yields the empty clause.*

The derivation of the empty clause formally proves that ϕ is unsatisfiable.

The iterative application of the resolution rule to a set of clauses can yield a complete solving procedure for SAT, giving rise to what is known as a variable elimination algorithm. This procedure is described in more detail in Section 3.2.

2.5 Tableaux for Boolean Propositional Logic

The tableau method is a systematic procedure for solving satisfiability problems, often viewed as a proof by case distinction. The tableau calculus for SAT provides a systematic procedure for determining the satisfiability of propositional formulas through semantic tree construction and branch closure analysis [37].

Using Smullyan's uniform notation [89], propositional formulas are grouped into conjunctive α-formulas and disjunctive β-formulas (Table 2.1). Together with double

Table 2.1: α-formulas and β-formulas.

α	α_1	α_2
$A \wedge B$	A	B
$\neg(A \vee B)$	$\neg A$	$\neg B$
$\neg(A \rightarrow B)$	A	$\neg B$

β	β_1	β_2
$A \vee B$	A	B
$\neg(A \wedge B)$	$\neg A$	$\neg B$
$A \rightarrow B$	$\neg A$	B

Table 2.2: Tableau expansion rules for SAT.

α	β	$\neg\neg A$	l
α_1	$\beta_1 \mid \beta_2$	A	$\neg l$
α_2			$\square$
α-rule	β-rule	$\neg$-rule	$\square$-rule

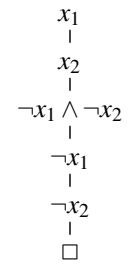


Figure 2.2: Completed SAT tableau for $\{x_1, x_2, \neg x_1 \wedge \neg x_2\}$.

negations $\neg\neg A$, this taxonomy allows SAT tableaux to break down complex formulas into their constituent parts using the expansion rules in Table 2.2.

A standard SAT tableau starts with a single branch containing the input formulas. Rules are applied until either every branch derives a contradiction (proving unsatisfiability) or at least one branch becomes saturated without contradictions (proving satisfiability). The contradiction rule ($\square$-rule) is applied to close a branch as soon as a literal and its negation appear simultaneously.

Example 2.5.1. *Figure 2.2 illustrates a classical SAT tableau for the unsatisfiable multiset $\phi = \{x_1, x_2, \neg x_1 \wedge \neg x_2\}$. The initial branch contains the three input formulas. Applying the α-rule to $\neg x_1 \wedge \neg x_2$ appends both $\neg x_1$ and $\neg x_2$ to the branch. Finally, the $\square$-rule detects the contradiction between x_1 and $\neg x_1$ (or alternatively x_2 and $\neg x_2$), closing the single branch with a $\square$. Since all branches are closed, ϕ is unsatisfiable.*

Extending the tableau framework to MaxSAT requires a fundamental shift in both execution strategy and rule design. First, expansion cannot stop upon finding a single

Table 2.3: Tableau expansion rules for MaxSAT.

α	β	$\neg\neg A$	l
$\square \mid \alpha_1$	$\beta_1 \mid \beta_2$	A	$\neg l$
$\mid \alpha_2$			$\square$
α-rule	β-rule	$\neg$-rule	$\square$-rule

contradiction; because the goal is to count all possible contradictions, rules must be applied until all branches are fully saturated. Second, to prevent under- or over-counting, formulas must act as consumable resources. Once a formula serves as a premise for a rule in a branch, it becomes *inactive* and cannot be reused. Thus, MaxSAT inference rules can be viewed as rewriting rules.

The classical α-rule presented in the previous section is unsound for MaxSAT because it does not preserve the minimum number of unsatisfied formulas. As seen in Figure 2.2, applying the standard α-rule to the instance $\{x_1, x_2, \neg x_1 \wedge \neg x_2\}$ decomposes the conjunction into $\neg x_1$ and $\neg x_2$. In a MaxSAT setting, this would allow the $\square$-rule to incorrectly derive two independent contradictions against x_1 and x_2, yielding a total cost of 2. However, the single optimal assignment (setting both x_1 and x_2 to true) only unsatisfies the original conjunction, meaning the true minimum cost is 1.

To remedy this overcounting problem, the complete MaxSAT tableau calculus introduces the branching α-rule shown in Table 2.3 [71]. This rule ensures that if an assignment unsatisfies α, the left branch is immediately penalized with a contradiction ($\square$) while freezing further decomposition of its subformulas. The right branch handles cases where α is satisfied, safely processing both α_1 and α_2. A conceptually parallel calculus was later proposed by Fiorino [45].

Soundness and completeness guarantee that once all branches are saturated under these resource constraints, the minimum number of derived contradictions among all branches exactly matches the minimum number of unsatisfied formulas in the input multiset [71].

Example 2.5.2. *Figure 2.3 details the step-by-step MaxSAT tableau construction for $\phi = \{x_1, x_2, \neg x_1 \wedge \neg x_2\}$, directly contrasting the SAT approach. Applying the modified branching α-rule yields the second configuration, saturating the left branch with a single $\square$. Two subsequent applications of the $\square$-rule to $\{x_1, \neg x_1\}$ and $\{x_2, \neg x_2\}$ on the right branch exhaust the active formulas, resulting in a final minimum contradiction count of 1 across the completed branches, matching the true optimal cost.*

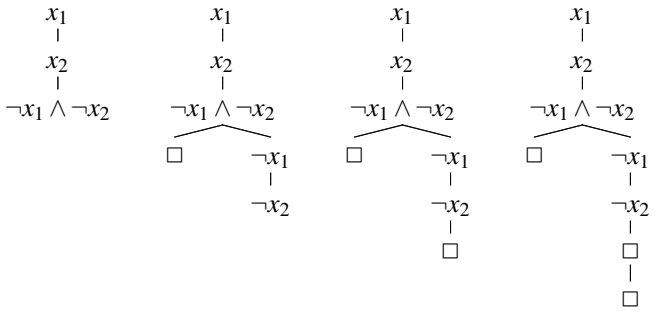


Figure 2.3: MaxSAT tableau construction for $\{x_1, x_2, \neg x_1 \wedge \neg x_2\}$.

Chapter 3

A Resolution Solver for Boolean MaxSAT

This chapter introduces the MaxSAT resolution rule and its extension to Weighted Partial MaxSAT. It then describes a variable elimination algorithm based on this resolution rule and presents MSElim, an efficient implementation of the algorithm that incorporates several features which can be turned on or off to assess their individual impact on performance. Finally, the chapter provides a comprehensive empirical evaluation of MSElim, both as a complete solver and as a Bounded Variable Elimination (BVE) preprocessor for state-of-the-art MaxSAT solvers.

3.1 Introduction

The classical resolution rule is sound and complete for SAT, preserving the satisfiability between its premises and conclusion [87]. However, this condition is insufficient in the context of MaxSAT. Since MaxSAT is an optimization problem, a sound inference rule must preserve the minimum number of unsatisfied clauses, not just satisfiability.

To illustrate that SAT resolution is unsound for MaxSAT, consider the multiset of clauses $C = \{x_1, \neg x_1 \vee x_2, \neg x_1 \vee \neg x_2, \neg x_1 \vee x_3, \neg x_1 \vee \neg x_3\}$. Under any optimal MaxSAT assignment (which sets x_1 to false), C contains one unsatisfied clause. However, if we apply classical unit propagation using the unit clause x_1, the multiset $C' = \{x_2, \neg x_2, x_3, \neg x_3\}$ is derived. Consequently, C' has a minimum of two unsatisfied clauses under any assignment. This increase in the number of unsatisfied clauses shows that unit propagation distorts the cost function, becoming unsound. Hence, this essential SAT solving technique cannot be applied to MaxSAT. This fundamental limitation led to the definition of a MaxSAT resolution rule that preserves the exact number of unsatisfied clauses for every assignment, yielding a sound and complete calculus for MaxSAT [24, 25, 61].

The MaxSAT resolution rule has attracted considerable attention within the proof complexity community, where foundational results have established that MaxSAT

resolution, when combined with structural rules, strictly dominates SAT resolution [21, 22, 23, 88]. However, a significant gap remains between theory and practice. While state-of-the-art branch-and-bound solvers use restricted variants of this rule [1, 27, 67, 68], no practical solver yet implements the full rule.

To bridge this gap, this chapter introduces a novel, exact, inference-based MaxSAT solver built directly upon the full resolution rule. Specifically, the underlying algorithm relies on a variable elimination process driven by MaxSAT resolution. While the resulting solver is not yet competitive with state-of-the-art MaxSAT solvers, which are predominantly based on the CDCL algorithm, the ultimate goal of this line of research is to develop efficient preprocessing techniques based on MaxSAT Bounded Variable Elimination (BVE). Here, we address the first necessary step: develop an efficient implementation of MaxSAT variable elimination. Moreover, we report initial experimental results that show significant speedups for specific problem families, confirming that BVE is a highly promising direction for MaxSAT solving.

The remainder of this chapter is structured as follows. Section 3.2 reviews the existing work on variable elimination algorithms for SAT. Section 3.3 presents the MaxSAT resolution rule, which forms the basis of the inference mechanism implemented in our solver, MSElim. Section 3.4 describes the architecture of MSElim. Finally, Section 3.5 evaluates the performance of MSElim using instances from the MaxSAT Evaluation. Finally, Section 3.6 concludes the chapter.

3.2 SAT Variable Elimination

The iterative application of the resolution rule to a set of clauses can yield a complete solving procedure for SAT. One way to approach this is through a variable elimination algorithm that iteratively eliminates a different variable at each step. Given a set of clauses C, let $C_x \subseteq C$ be the clauses containing x and $C_{\neg x} \subseteq C$ those containing $\neg x$. Eliminating a variable x consists first in resolving every clause of C_x against every clause of $C_{\neg x}$ on x, discarding the tautologies, and obtaining a new formula C'. Then, the original clauses $C_x \cup C_{\neg x}$ are removed from C', obtaining a formula that no longer contains occurrences of variable x but has the same satisfiability as C. The key point is that satisfiability is preserved because the formula C' is saturated with respect to x, according to the following definition.

Definition 3.2.1. *A set C of clauses is* saturated *with respect to variable x if, for every pair of resolvable clauses $x \vee A \in C_x$ and $\neg x \vee B \in C_{\neg x}$, either $A \vee B$ is a tautology or $A \vee B \in C$. In other words, C is saturated if any newly generated resolvent is logically implied by an existing clause in C.*

A set C' of clauses is a saturation of *C with respect to x if C' is saturated with respect to x and $C \vdash_x C'$; that is, C' is obtained from C by finitely many applications of the resolution rule, each resolving on x.*

This is known as the Davis–Putnam elimination step [38], which yields a complete SAT algorithm.

Algorithm 3.1 describes the SAT solving process based on iterative variable elimination. This algorithm eliminates all variables in a CNF formula C in some order; for

Algorithm 3.1: An exact variable elimination algorithm for SAT

Input: C: a set of clauses.
Output: (s, I): s is *UNSAT* if the formula is unsatisfiable, otherwise s is *SAT* and I an assignment that satisfies all clauses in C.

$C_0 \leftarrow C$;
for $i \leftarrow 1$ **to** n **do**
  $C'_i \leftarrow saturation(C_{i-1}, x_i)$;
  $\langle C_i, D_i \rangle \leftarrow partition(C'_i, x_i)$;
$I \leftarrow \emptyset$;
if $\square \in C_n$ **then**
  $s =$UNSAT;
else
  $s =$SAT;
  for $i \leftarrow n$ **downto** 1 **do**
    $I \leftarrow I \cup [x_i \mapsto extension(x_i, I, D_i)]$;
return (s, I)

simplicity, we assume w.l.o.g. the lexicographical order: $x_1, \ldots, x_n$. The algorithm generates three sequences of sets of clauses: $C_0, \ldots, C_n$; $C'_1, \ldots, C'_n$; and $D_1, \ldots, D_n$, where $C = C_0$; for $i = 1, \ldots, n$, C'_i is a saturation of C_{i-1} with respect to x_i; C_i is the set of clauses not containing occurrences of variables $x_1 \ldots, x_i$; and $D_i = C'_{ix} \cup C'_{i\neg x}$. If the empty clause is derived during the process, C is unsatisfiable. Otherwise, a model I of the formula can be reconstructed iteratively by visiting the sets D_i in reverse elimination order and, at each step, assigning variable x_i a value such that I satisfies D_i. This can be done efficiently, since the saturation property guarantees that I satisfies C'_{ix} or $C'_{i\neg x}$, regardless of the value assigned to x [38].

The major performance drawback of Algorithm 3.1 is that each variable elimination step can cause a quadratic growth in the formula size. Consequently, in practice, this approach is highly inefficient and cannot compete with state-of-the-art SAT solvers based on the Conflict-Driven Clause Learning (CDCL) algorithm. However, restricting this elimination process to a heuristically chosen subset of variables has proven to be a highly effective preprocessing and inprocessing technique for simplifying formulas before and during the search in CDCL solvers [18, 75]. This bounded approach is known as Bounded Variable Elimination (BVE). Specifically, before eliminating a variable x_i, the net increase in formula size is computed; the elimination is then executed only if this increase satisfies a predefined threshold or criterion [39].

The effectiveness of BVE is significantly enhanced when complemented with other inference techniques, such as unit propagation, subsumption, and self-subsuming resolution, as described below.

Unit Propagation Unit Propagation (UP) is the iterated application of the *Unit Clause* (UC) *rule* (also referred to as the *one-literal rule*) until an empty clause is derived or there are no unit clauses left. If a clause is unit, then its literal must be as-

signed the value *true*. If ℓ is a unit clause in a CNF formula C, UC consists in deleting all the clauses of C with literal ℓ and removing all the occurrences of literal $\neg\ell$. Modern solvers implement UP using the *two-watched-literal* scheme introduced with the Chaff solver [84, 99]. In these implementations, the concept of unit clause is adapted to the co-existence of a partial assignment, i.e. a clause is unit if it has all its literals falsified but one, which is unassigned. The idea is to watch two non-false literals in each clause. While both watches are non-false, the clause can neither propagate nor be conflicting so that it can be ignored. A clause is therefore visited only when one of its watched literals becomes false. To know which clauses to visit, the solver keeps a *watch list* for each literal ℓ. This list contains the clauses in which ℓ is currently watched.

Subsumption A clause A subsumes a clause B when every literal of A also appears in B, that is, seeing clauses as sets of literals, $A \subseteq B$. In that case, B is implied by A and can be deleted, since any assignment that satisfies A also satisfies B. In practice, given a set of clauses C, when checking possible subsumption relations between a clause $A \in C$ and the other clauses in $C' = C \setminus \{A\}$, the check can run in two directions: *forward subsumption* asks whether a A is subsumed by some clause in C' (hence A can be removed from C), while *backward subsumption* asks whether A subsumes any clause in C' (and hence such clauses can be removed from C). The variable elimination process generates a number of new clauses after a variable has been eliminated, and such clauses are good candidates to be checked for forward and backward subsumption with the other clauses in the formula.

Self-subsuming resolution Let clauses A and B be resolvable on a variable x. If their resolvent subsumes some of the premises, these premises can be removed. Self-subsuming resolution can either be applied during a resolution rule application between two clauses A and B during the elimination of variable x, or when checking the subsumption relations between two clauses A and B.

3.3 MaxSAT Resolution Rule

The MaxSAT resolution rule can be considered an extension of the classical resolution rule that preserves the number of unsatisfied clauses from the premises to the conclusions for any assignment [24, 60]. Moreover, the MaxSAT resolution rule is sound and complete for MaxSAT [24, 25]. Unlike classical resolution, which adds the conclusions to the existing set, the MaxSAT resolution rule replaces the premises with the conclusions.

Definition 3.3.1. *Given two clauses* $x \vee a_1 \vee \cdots \vee a_s$ *and* $\neg x \vee b_1 \vee \cdots \vee b_t$*, the MaxSAT*

resolution rule replaces them with the following multiset of clauses:

$$\begin{array}{l} x \vee a_1 \vee \cdots \vee a_s \\ \neg x \vee b_1 \vee \cdots \vee b_t \\ \hline\hline a_1 \vee \cdots \vee a_s \vee b_1 \vee \cdots \vee b_t \\ x \vee a_1 \vee \cdots \vee a_s \vee \neg b_1 \\ x \vee a_1 \vee \cdots \vee a_s \vee b_1 \vee \neg b_2 \\ \cdots \\ x \vee a_1 \vee \cdots \vee a_s \vee b_1 \vee \cdots \vee b_{t-1} \vee \neg b_t \\ \neg x \vee b_1 \vee \cdots \vee b_t \vee \neg a_1 \\ \neg x \vee b_1 \vee \cdots \vee b_t \vee a_1 \vee \neg a_2 \\ \cdots \\ \neg x \vee b_1 \vee \cdots \vee b_t \vee a_1 \vee \cdots \vee a_{s-1} \vee \neg a_s \end{array}$$

The tautologies among the conclusions are removed, and repeated literals within a clause are collapsed into a single occurrence. Besides the conclusion in which the variable is resolved, the rule also produces additional clauses, called compensation clauses. We say that the rule resolves the variable x.

The first conclusion is the classical resolvent and does not contain the variable x. The remaining conclusions are called compensation clauses; each one contains one of the original premises as a subclause and ensures that the number of unsatisfied clauses is preserved.

In the sequel, we often use the following compact representation of the rule:

$$\begin{array}{l} x \vee A \\ \neg x \vee B \\ \hline\hline A \vee B \\ x \vee A \vee \neg B \\ \neg x \vee \neg A \vee B \end{array}$$

where $A = a_1 \vee \cdots \vee a_s$ and $B = b_1 \vee \cdots \vee b_t$ are disjunctions of literals. Notice that $\neg A$ and $\neg B$ are not in clausal form. Their MaxSAT clausal forms are $\{\neg a_1,\ a_1 \vee \neg a_2,\ \ldots,\ a_1 \vee \cdots \vee a_{s-1} \vee \neg a_s\}$ and $\{\neg b_1,\ b_1 \vee \neg b_2,\ \ldots,\ b_1 \vee \cdots \vee b_{t-1} \vee \neg b_t\}$, respectively, as shown in [70, 72]. Bonet et al. [24, 25] proved that the MaxSAT resolution rule is sound and complete.

The number of conclusions is bounded by the number of literals in the premises; when the two premises share literals, some compensation clauses become tautologies and are removed. In particular, when $A = B$, all compensation clauses are tautological, and the only non-tautological conclusion is A itself.

Example 3.3.2. *Consider the multiset of clauses $C = \{x_1, \neg x_1 \vee x_2, \neg x_2 \vee x_3, \neg x_3\}$ to illustrate how variable saturation works. We begin by saturating with respect to x_1: resolving x_1 with $\neg x_1 \vee x_2$ yields the resolvent x_2 and the compensation clause $x_1 \vee \neg x_2$, producing $\{x_2, x_1 \vee \neg x_2, \neg x_2 \vee x_3, \neg x_3\}$. Since $x_1 \vee \neg x_2$ is the only clause containing x_1 and there is no clause with $\neg x_1$, the multiset is saturated with respect to x_1. Setting aside the clause containing x_1, we obtain $C_1 = \{x_2, \neg x_2 \vee x_3, \neg x_3\}$. Next, we saturate*

C_1 with respect to x_2: resolving x_2 with $\neg x_2 \vee x_3$ yields the resolvent x_3 and the compensation clause $x_2 \vee \neg x_3$, producing $\{x_3, x_2 \vee \neg x_3, \neg x_3\}$. Again, no further resolution on x_2 is possible, so we set aside $x_2 \vee \neg x_3$ and obtain $C_2 = \{x_3, \neg x_3\}$. Finally, resolving x_3 with $\neg x_3$ yields $\square$ (the empty clause). Since all variables have been saturated, the minimum number of unsatisfied clauses in C is 1.

3.3.1 Weighted Partial MaxSAT Resolution

The resolution rule introduced above is for unweighted MaxSAT. We now extend it to Weighted MaxSAT. In this context, each clause is assigned a positive weight, and the rule operates on two weighted premises, $(x \vee A, u)$ and $(\neg x \vee B, w)$, as follows:

$$\begin{array}{l} (x \vee A, u) \\ (\neg x \vee B, w) \\ \hline\hline (A \vee B, \min(u,w)) \\ (x \vee A \vee \neg B, \min(u,w)) \\ (\neg x \vee \neg A \vee B, \min(u,w)) \\ (x \vee A \vee \neg B, u - \min(u,w)) \\ (\neg x \vee \neg A \vee B, w - \min(u,w)) \end{array}$$

Any conclusion with a weight of zero is discarded. Intuitively, a clause with weight w behaves as w identical copies of that clause, so a single application of the weighted rule accounts for $\min(u,w)$ applications of its unweighted counterpart at once.

When some clauses are declared hard, we are in the partial setting. A hard clause is assigned an infinite weight and cannot be falsified by any feasible assignment. Three cases arise, depending on which premises are hard:

$$\begin{array}{l} (x \vee A, u) \\ (\neg x \vee B, \infty) \\ \hline\hline (A \vee B, u) \\ (x \vee A \vee \neg B, u) \\ (\neg x \vee B, \infty) \end{array} \qquad \begin{array}{l} (x \vee A, \infty) \\ (\neg x \vee B, w) \\ \hline\hline (A \vee B, w) \\ (x \vee A, \infty) \\ (\neg x \vee \neg A \vee B, w) \end{array} \qquad \begin{array}{l} (x \vee A, \infty) \\ (\neg x \vee B, \infty) \\ \hline\hline (A \vee B, \infty) \\ (x \vee A, \infty) \\ (\neg x \vee B, \infty) \end{array}$$

In each case, the hard premise is retained in the conclusions to ensure its constraint remains enforced. When two hard unit clauses resolve to produce the empty clause, the result is $(\square, \infty)$, denoted by $\blacksquare$. This signifies that the hard part of the formula is unsatisfiable, rendering the instance infeasible.

Example 3.3.3. *Consider the multiset $\{(x_1,3), (\neg x_1 \vee x_2, 2)\}$. Applying the weighted rule yields $\{(x_2,2), (x_1 \vee \neg x_2, 2), (x_1,1)\}$. If the first clause is hard, the multiset $\{(x_1,\infty)$ $, (\neg x_1 \vee x_2, 2)\}$ resolves to $\{(x_2,2), (x_1,\infty)\}$. When both clauses are unit, the multiset $\{(x_1,\infty), (\neg x_1, 2)\}$ produces $\{(\square, 2), (x_1,\infty)\}$. Finally, $\{(x_1,\infty), (\neg x_1,\infty)\}$ derives $\{\blacksquare\}$: the hard part is unsatisfiable, so no optimal assignment exists.*

3.4 MSElim: a Variable Elimination Solver for MaxSAT

This section presents MSElim, a complete reimplementation of the variable elimination approach to MaxSAT solving introduced in my master's thesis [95]. While the feasibility of exact variable elimination based on the MaxSAT resolution rule was explored in that earlier work, the method has received limited attention as a practical solving technique.

MSElim solves Weighted Partial MaxSAT by repeatedly selecting a variable, resolving it away using MaxSAT resolution, and eliminating it from the working formula. The main difficulty of MaxSAT variable elimination is that the resolution rule produces not only a resolvent but also compensation clauses, which may cause the formula to grow rapidly. Our new implementation is designed to keep this growth under control. It also combines variable elimination with other powerful techniques such as unit propagation and clause subsumption. Although this combination has become standard in SAT solvers, it has never been explored in MaxSAT variable elimination, where additional challenges arise. Besides, we show that specific MaxSAT inference techniques can be applied together with clause subsumption and unit propagation.

This section is organized as follows. Section 3.4.1 presents the generic variable elimination procedure for MaxSAT described in [24]. Section 3.4.2 explains how formulas are represented in our implementation. Section 3.4.3 introduces heuristics for selecting the next variable to eliminate. Section 3.4.4 describes two alternative implementations of the saturation step. Section 3.4.5 presents the simplification techniques applied in combination with the elimination process.

Recall that, in the context of Weighted Partial MaxSAT, clauses are defined as pairs of the form (A, w), where A is a disjunction of literals, and w is either a non-negative value (for soft clause) or infinite (for hard clause). However, in several of the explanations below, the weight plays no role. For simplicity, and whenever the meaning is clear from the context, we will refer to weighted clauses simply as clauses. Likewise, clause weights will be omitted when they are irrelevant.

3.4.1 A Generic Variable Elimination Algorithm

In this section, we show that MaxSAT and its weighted and partial variants can also be solved with a variable elimination algorithm similar to Algorithm 3.1 for SAT. Algorithm 3.2 presents the pseudo-code for the most general case, Weighted Partial MaxSAT. At a high level, the pseudo-code differences with Algorithm 3.1 are minor, only introduced by the difference in the problem definition and the kind of result (satisfiability versus optimization). However, when there are soft clauses, the notion of saturation differs from that introduced in Definition 3.2.1. Moreover, a saturation in MaxSAT must be computed using the MaxSAT resolution rule instead of the resolution rule. We first present the notion of saturation for unweighted MaxSAT.

Definition 3.4.1. *A multiset C of clauses is said to be* saturated with respect to variable *x if, for every pair of clauses $x \vee A \in C_x$ and $\neg x \vee B$ in $C_{\neg x}$, there exists a literal*

Algorithm 3.2: An exact variable elimination algorithm for Weighted Partial MaxSAT

Input: C: a multiset of weighted clauses.
Output: (m, I): if there exists an assignment that satisfies all hard clauses, I is one that minimizes the sum of weights of falsified soft clauses and m is this sum. Otherwise, $m = \infty$.

$C_0 \leftarrow C$;
for $i \leftarrow 1$ **to** n **do**
  $C'_i \leftarrow saturation(C_{i-1}, x_i)$;
  $\langle C_i, D_i \rangle \leftarrow partition(C'_i, x_i)$;
$I \leftarrow \emptyset$;
if $(\square, \infty) \in C_n$ **then**
  $m = \infty$;
else
  $m = \sum_{(\square, w) \in C_n} w$;
  for $i \leftarrow n$ **downto** 1 **do**
    $I \leftarrow I \cup [x_i \mapsto extension(x_i, I, D_i)]$;
return (m, I)

ℓ such that ℓ appears in A and $\neg\ell$ appears in B. In other words, its resolvent is a tautology.

A multiset C' is a saturation of *C with respect to x if C' is saturated with respect to x and $C \vdash_x C'$; that is, C' is obtained from C by finitely many applications of the MaxSAT resolution rule, each resolving on x.*

The intuition behind this definition is straightforward: once a multiset is saturated with respect to x, every remaining pair of clauses $x \vee A$ and $\neg x \vee B$ produces a tautological resolvent $A \vee B$, so any further application of the MaxSAT resolution rule on x would only generate compensation clauses without introducing new variable-free clauses.

Since the weights of the clauses do not play any role in the notion of saturation, the previous definition is also valid for Weighted MaxSAT. However, for Weighted Partial MaxSAT, the definition of saturation combines Definitions 3.2.1 and 3.4.1.

Definition 3.4.2. *A multiset C of clauses is saturated with respect to x if two conditions hold:*

1. *For every pair of resolvable hard clauses $x \vee A \in C_x$, $\neg x \vee B \in C_{\neg x}$, there is a clause $E \in C$ such that $E \subseteq A \cup B$.*

2. *For every pair of clauses $x \vee A \in C_x$ and $\neg x \vee B$ in $C_{\neg x}$ such that at least one of them is soft, there exists a literal ℓ such that ℓ appears in A and $\neg\ell$ appears in B. In other words, its resolvent is a tautology.*

A multiset C' of clauses is a saturation of *C with respect to x if C' is saturated with*

respect to x and $C \vdash_x C'$; that is, C' is obtained from C by finitely many applications of the MaxSAT resolution rule, each resolving on x.

Example 3.4.3. *To illustrate how Algorithm 3.2 operates, we consider the multiset $C = \{x_1 \vee \neg x_3,\ \neg x_1 \vee x_2,\ x_3,\ \neg x_2\}$, where all clauses are soft and have the same weight. We set $C_0 = C$ and eliminate variables following the lexicographical order: x_1, x_2, x_3.*

***Iteration** $i = 1$ **(eliminate** x_1**):** In C_0 two clauses mention x_1: the positive clause $x_1 \vee \neg x_3$ and the negative clause $\neg x_1 \vee x_2$. Applying MaxSAT resolution to this pair produces three clauses: the resolvent $x_2 \vee \neg x_3$ and the two compensation clauses $x_1 \vee \neg x_2 \vee \neg x_3$ and $\neg x_1 \vee x_2 \vee x_3$. The remaining clauses x_3 and $\neg x_2$ are untouched, so the saturation of C_0 with respect to x_1 is $\{x_2 \vee \neg x_3,\ x_1 \vee \neg x_2 \vee \neg x_3,\ \neg x_1 \vee x_2 \vee x_3,\ x_3,\ \neg x_2\}$. After partitioning, we obtain $C_1 = \{x_2 \vee \neg x_3,\ x_3,\ \neg x_2\}$ (the clauses free of x_1) and $D_1 = \{x_1 \vee \neg x_2 \vee \neg x_3,\ \neg x_1 \vee x_2 \vee x_3\}$ (the clauses that still mention x_1, stored for later use).*

***Iteration** $i = 2$ **(eliminate** x_2**):** Within C_1 the only resolution pair on x_2 is $(x_2 \vee \neg x_3)$ and $(\neg x_2)$. Resolution yields the resolvent $\neg x_3$ together with the compensation clause $\neg x_2 \vee x_3$. Combined with the clause x_3 that does not contain x_2, the full saturation is $\{x_3,\ \neg x_3,\ \neg x_2 \vee x_3\}$. Partitioning gives $C_2 = \{x_3,\ \neg x_3\}$ and $D_2 = \{\neg x_2 \vee x_3\}$.*

***Iteration** $i = 3$ **(eliminate** x_3**):** Resolving x_3 with $\neg x_3$ in C_2 produces the empty clause $\square$ (soft clause). Hence $C_3 = \{\square\}$ and $D_3 = \emptyset$. Since $m = |C_3| = 1$, every assignment falsifies at least one clause in C.*

***Backward pass (constructing an optimal assignment):** We build the assignment I by extending it one variable at a time from x_3 down to x_1, selecting each value so that it satisfies all clauses in D_i.*

Because $D_3 = \emptyset$, the value of x_3 is unconstrained; we set $x_3 \mapsto$ false. Under $x_3 =$ false, setting $x_2 \mapsto$ false satisfies the clause in $D_2 = \{\neg x_2 \vee x_3\}$. Finally, under $x_2 = x_3 =$ false, setting $x_1 \mapsto$ false satisfies both clauses in D_1.

The algorithm returns $m = 1$ with the optimal assignment $x_1 \mapsto$ false, $x_2 \mapsto$ false, $x_3 \mapsto$ false. One can verify that this assignment satisfies three of the four clauses in C, leaving only x_3 unsatisfied.

A noteworthy property is that the saturation of a given multiset with respect to a variable is not necessarily unique. Different orderings for resolving pairs of clauses can lead to distinct saturated multisets. As an illustration, the multiset of unweighted soft clauses $\{x_1, \neg x_1 \vee x_2, \neg x_1 \vee x_3\}$ admits two saturations with respect to x_1: one yielding $\{x_2, \neg x_2 \vee x_3, x_1 \vee \neg x_2 \vee \neg x_3, \neg x_1 \vee x_2 \vee x_3\}$ and another yielding $\{x_3, x_2 \vee \neg x_3, x_1 \vee \neg x_2 \vee \neg x_3, \neg x_1 \vee x_2 \vee x_3\}$. This non-uniqueness does not affect correctness: the completeness theorem guarantees that the number of derived empty clauses remains the same, regardless of which saturation is computed [25].

Finally, note that for Weighted Partial MaxSAT, the variants of the MaxSAT resolution rule presented in Section 3.3.1 must be applied. Rather than providing a full example, we illustrate this with a single saturation step.

Example 3.4.4. *To illustrate the weighted partial variant of the rule, we consider the multiset $C = \{(\neg x_1 \vee \neg x_2,\ 1),\ (\neg x_1 \vee \neg x_3,\ 2), (x_1 \vee \neg x_2,\ \infty)\}$. We set $C_0 = C$ and focus exclusively on eliminating variable x_1.*

First rule application: $(x_1 \vee \neg x_2,\ \infty)$ *and* $(\neg x_1 \vee \neg x_2,\ 1)$ *:*

$$\begin{array}{l} (x_1 \vee \neg x_2,\ \infty) \\ (\neg x_1 \vee \neg x_2,\ 1) \\ \hline\hline (\neg x_2,\ 1) \\ (x_1 \vee \neg x_2,\ \infty) \end{array}$$

Second rule application: $(x_1 \vee \neg x_2,\ \infty)$ *and* $(\neg x_1 \vee \neg x_3,\ 2)$*:*

$$\begin{array}{l} (x_1 \vee \neg x_2,\ \infty) \\ (\neg x_1 \vee \neg x_3,\ 2) \\ \hline\hline (\neg x_2 \vee \neg x_3,\ 2) \\ (x_1 \vee \neg x_2,\ \infty) \\ (\neg x_1 \vee x_2 \vee \neg x_3,\ 2) \end{array}$$

After these two applications of the rule, the active multiset is $C = \{(\neg x_2,\ 1),\ (\neg x_2 \vee \neg x_3,\ 2), (x_1 \vee \neg x_2,\ \infty), (\neg x_1 \vee x_2 \vee \neg x_3,\ 2)\}$ *which is saturated with respect to* x_1*. The only clause containing* x_1 *is the hard clause* $(x_1 \vee \neg x_2,\ \infty)$*, and the only clause containing* $\neg x_1$ *is the soft clause* $(\neg x_1 \vee x_2 \vee \neg x_3,\ 2)$*. Since one of them is soft, only condition 2 of Definition 3.4.2 applies; their resolvent cancels the literal* x_2 *and is therefore a tautology, so no further application to* x_1 *produces a new variable-free clause, and the saturation stops.*

3.4.2 Formula Representation

The choice of data structures for formula storage forms the basis of our variable elimination implementation. In this subsection, we present the building blocks from the bottom up—starting with variables and literals, followed by hard and soft clauses—and describe for each how it is stored and how that choice is used during elimination.

Our representation of these elements is standard and adopted from the MiniSAT solver [40, 90]. We also incorporate the additional data structures required by MSElim: the clause allocator, the assignment trail, watch lists, and occurrence lists.

The part of the representation that is specific to the weighted MaxSAT setting is the treatment of soft unit clauses, which are not stored as clauses but as weights in an array indexed by literal.

Literals and Variables

We adopt the variable and literal encoding of MiniSat [40], summarized in Table 3.1. A variable is a plain integer in the range $0, \ldots, n-1$: variables are introduced in sequence and serve directly as array indices, so any information about the variable is stored in a flat array with one entry per variable. A literal packs a variable and its sign into a single integer $\ell = 2x + s$, with $s = 0$ for the positive literal and $s = 1$ for the negative one. Eén and Sörensson [40] describe this packed integer as "a 'small' integer suitable for array indexing": the literal is its own index. Three properties follow that the rest of the code relies on. First, negation is a single bit flip ($\ell \mapsto \ell \oplus 1$): since the sign is the lowest bit,

Field	Expression	Meaning
Variable	$v \in \{0,1,2,\dots\}$	a variable is an index
Literal	$x = 2v + s$	packs variable v and sign bit $s \in \{0,1\}$
Negation	$x \mapsto x \oplus 1$	flip the low bit to swap polarity
Variable of a literal	$\text{var}(\ell) = x \gg 1$	drop the sign bit
Sign of a literal	$\text{sign}(\ell) = x \,\&\, 1$	low bit: 0 positive, 1 negative

Table 3.1: Literal encoding in the implementation of MSElim.

Clause type	Where it is stored
size ≥ 2 (hard or soft)	clause allocator (`ca`), listed in `clauses`
unit hard	assignment trail (`trail`), with value in `assigns`
unit soft	array indexed by literal (`unit_soft`)
empty hard	none — the instance is unsatisfiable, the process will finish immediately
empty soft	global cost counter (`empty_clauses`)

Table 3.2: Clause representation in MSElim, depending on its size and kind (hard or soft).

there is no polarity test and no branch for the processor to predict. Second, the literal can be used directly as the index of any literal indexed array; there is no hashing and no lookup table, because the integer value of the literal is already the position of its entry. Third, a literal and its negation are placed at consecutive positions ($2x$ and $2x+1$), so both polarities of a variable lie next to each other in any literal-indexed array, which means the two are read together from the same region of memory. These look like small details, but resolution negates literals and indexes by literal constantly, so they apply to the hottest path of the procedure.

Each variable is assigned a truth value (a three-valued `lbool`: true, false, or unassigned), where the third value marks that the variable has not yet been given a value. Beyond its truth value, our elimination procedure assigns a few additional attributes to each variable, such as a flag indicating whether the variable has already been eliminated or skipped (will not be eliminated at all) in the BVE scenario.

Hard and soft clauses

Each clause stores a weight, and a reserved maximal value ∞ marks the clause as hard; any smaller weight is the cost of falsifying a soft clause. So a single field tells us both the cost and the type of a clause. Unit and empty clauses are not kept as clauses at all, while clauses of two or more literals are kept in a general clause database. Table 3.2 summarises this.

There is a global integer counter that represents the cost of the formula derived so far, that is, the sum of the weights of the derived empty soft clauses.

A unit hard clause (ℓ, ∞) asserts that a literal must be true, which is the same statement as forcing that literal true; we therefore store it not as a clause but directly on the assignment *trail*. The trail is a standard structure in conflict-driven solvers: a sequence of the literals currently set to true, in the order they were assigned. In a CDCL solver, the trail is partitioned into decision levels and is unwound on backtracking [81, 79]; our solver never backtracks, so the trail only ever grows.

A unit soft clause of weight w and a literal ℓ is not stored as a clause but as a number. In particular, we add w to the slot for ℓ in a vector with two slots per variable, one for each polarity. If we add a soft unit on ℓ while its negation $\neg\ell$ already carries weight, the two cannot both be satisfied: any assignment falsifies one of them, so at least the smaller of the two weights must be paid no matter what. We therefore move that smaller weight into the cost counter at once and subtract it from both slots, which leaves at most one polarity of the variable with weight. This process can be seen as an application of the MaxSAT resolution rule to the soft premises x and $\neg x$.

Soft and hard clauses containing two or more literals are represented as real clauses and are stored in the shared clause database. The main fields associated with each clause are the following:

size: number of literals of the clause.

data: vector of literals of the clause.

removed: flag indicating if the clause has been removed.

mark_strengthen and mark_subsume: flags indicating whether the clause has already been added to the list of clauses to be strengthened or to be used for backward subsumption (Section 3.4.5).

abstraction: this is a 64-bit signature that records the indices of the variables (modulo 64) that the clause contains. For instance, a clause containing x_0 will set bit number 0 to 1. Note that the abstraction cannot differentiate if the clause contains literal x_0 or $\neg x_0$, and neither whether it contains variable x_0, or x_{64}, or x_{128}, ..., or many of them. The abstraction is crucial to speed up the subsumption process (Section 3.4.5), and is also used by one of the variable selection heuristics (Section 3.4.3).

So far, we have seen where each element is stored and how it is stored. However, during saturation and unit propagation, we need information about the set of clauses where a literal occurs. We keep two indices from literals to clauses. The first index is the occurrence lists. In order to eliminate a variable x, we need to identify the clauses containing literal x and the clauses containing literal $\neg x$. For this, we keep occurrence lists, which map each literal ℓ to all clauses containing ℓ. The second index is the watch lists, which record clauses under only two of its literals. Watch lists are used for unit propagation and for forward subsumption. We maintain separate watch lists for hard and soft clauses. On the one hand, as explained later, the implementation of forward subsumption checks only the watches of hard clauses. On the other hand, the two behave differently during unit propagation. Falsifying a hard clause means that the formula is unsatisfiable, and finding a unit hard clause triggers a new assignment in the trail. However, falsifying a soft clause results in an increase in the cost equal to its weight, while any detected unit soft clause is removed from the clause database, and the weight of the unassigned literal is updated in the unit soft literal representation array.

Example 3.4.5. *Consider the weighted soft clauses* $(a \vee b \vee c, 5)$ *and* $(\neg a, 3)$*, and the hard clauses* $(\neg b, \infty)$ *and* $(\neg c, \infty)$*. Setting b and c to false reduces the first clause to the soft clause* $(a, 5)$*. Adding it to the unit soft array meets the existing* $(\neg a, 3)$*: since* (a) *and* $(\neg a)$ *cannot hold simultaneously, the smallest weight,* 3*, must be paid, leaving* $(a, 2)$*. The resulting simplified instance has 3 empty clauses (or an empty clause with weight* 3*), the unit soft clause* $(a, 2)$ *and the two initial hard clauses. This is correct: once* $b = c =$ *false, the best choice is* $a =$ *true at cost* 3*.*

3.4.3 Variable Selection Heuristics

The order of variable elimination is a critical factor determining performance. Because MaxSAT variable elimination can cause formulas to explode in size, eliminating cheap variables first keeps the formula small for longer. This, in turn, reduces the computational cost of subsequent eliminations, as each operation processes fewer and shorter clauses. A pure variable represents the extreme case: lacking clauses of one polarity, the clauses with the other polarity can be removed for free. Prioritizing these variables while delaying expensive ones (or avoiding them entirely, as in BVE) prevents the intermediate formula from blowing up.

It is important to clarify the exact impact of the heuristic. While it affects the execution speed of the elimination process and the total number of variables eliminated, it has no bearing on the final result, as resolution-based elimination is sound regardless of the elimination order.

We have implemented four different heuristics to select the variables to be eliminated. The first one (Heuristic 0) simply picks the variables in lexicographic order (i.e., from 0 to $n_{Vars}-1$) and serves as a baseline to evaluate the remaining three. The other heuristics use a min-heap (reusing MiniSAT's implementation) to select the variable with the minimum score each time we need to pick a new variable to eliminate. When computing any of these heuristics, we can check whether a literal has become pure at no extra cost by reading the size of its occurrence lists. If either of the two lists is empty, the variable is pure, and we set its score to 0 so that the heap prioritizes it. The heuristics are defined as follows:

Heuristic 1 (degree): The score in this heuristic is the number of non-unit clauses in which the variable occurs, counting both polarities. It is computed in constant time by adding the sizes of the occurrence lists of the variable's two literals, without scanning the clauses themselves. We refer to this metric as the clause degree of the variable. Since the goal is to select the cheapest variable for elimination, this heuristic prioritizes the variable with the lowest clause degree. Figure 3.1 illustrates this concept: variable x occurs in two clauses while y occurs in five, making x the cheaper alternative.

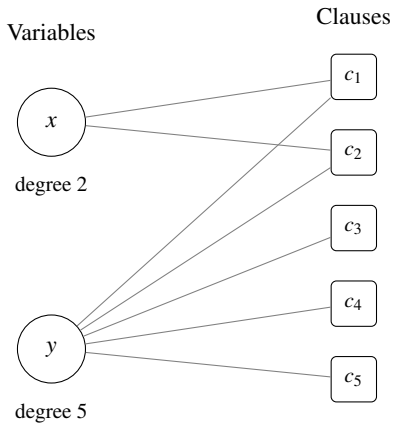


Figure 3.1: Bipartite graph linking variables to the clauses that contain them.

Heuristic 3 (full neighbourhood): This heuristic scores a variable by the size of its neighbourhood, which is the number of distinct variables, other than the variable itself, that occur in some clause together with it. Equivalently, it is the degree of the variable in the variable interaction graph of the formula, where two variables are joined whenever they share a clause. Computing it means walking every literal of every clause the variable appears in, which makes this the most time-consuming of our heuristics in terms of computation of the scores. Figure 3.2 contains a variable interaction graph, where we show the idea of full neighbourhood. Each node is a variable, and an edge connects two variables whenever they co-occur in a clause. In this example, x_0 appears together with x_3, x_4, x_5, and x_6, so its neighbourhood size is 4. Variables x_1 and x_2 each share clauses with only two other variables, giving them a neighbourhood size of 2.

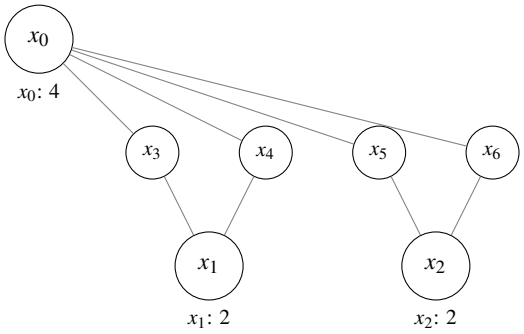


Figure 3.2: Connected variable interaction graph for the clause set $\{x_0 \vee x_3,\ x_0 \vee x_4,\ x_0 \vee x_5,\ \neg x_0 \vee x_6,\ x_1 \vee x_3,\ \neg x_1 \vee x_4,\ x_2 \vee x_5,\ \neg x_2 \vee x_6\}$.

Heuristic 2 (neighbourhood abstraction): This is a cheaper version of Heuristic 3. It uses the 64-bit abstractions of the clauses to approximate the size of a variable's

neighbourhood. More precisely, it computes the union of the abstractions of the clauses containing a variable, without needing to iterate over the literals of every clause (unlike heuristic 3). However, two distinct neighbours can collide on the same bit, hence this process yields an underestimation of the real neighbourhood size. In Figure 3.3, we show the score computation of x_0. Two different clauses contain x_0, and their abstractions are identical. However, if we look at their real neighbourhoods, we see that x_0 has 3 neighbours in C_2. The reason both clauses have the same abstraction but different neighbourhood sizes is that, in the 64-bit representation, x_{65} and x_{66} share the same bit as x_1 and x_2, respectively.

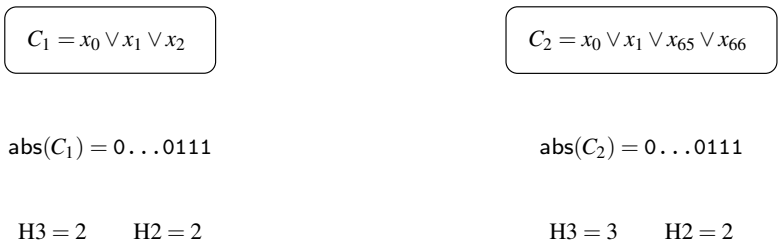


Figure 3.3: Bit-collision underestimation of the clause abstraction when scoring x_0.

3.4.4 Variable Saturation

Once we have heuristically chosen the variable x to be eliminated, we must saturate the set of clauses C with respect to x; that is, we perform the *saturation* process of Algorithm 3.2. In SAT solvers, where only hard clauses occur, this step is straightforward to implement (recall Definition 3.2.1): given the sets C_x and $C_{\neg x}$ of clauses that contain x and $\neg x$, respectively, two simple nested loops iterate over all pairs of clauses $A \in C_x$ and $B \in C_{\neg x}$ while adding their resolvents to C. In fact, in BVE for SAT, some implementations first simulate this process without actually generating new clauses to decide whether a variable should be eliminated based on the resulting increase in formula size.

However, saturating a formula C in the MaxSAT case, using MaxSAT resolution, is more challenging. The main difficulty is that the sets C_x and $C_{\neg x}$ are not static, since applying the rule removes premises from these sets and adds new compensation clauses. Moreover, it is required to generate the compensation clauses even just to anticipate the growth of the formula. Two different implementations will be proposed later. We first present a general high-level algorithm to solve the *saturation* process for Weighted Partial MaxSAT. It is described in Algorithm 3.3. It starts by obtaining the sets C_x and $C_{\neg x}$ that will be iterated for saturation, and generates three sets of clauses: C'_x, $C'_{\neg x}$ and L. The first two sets are the sets of clauses containing x and $\neg x$ after saturation is completed; they are updated as the algorithm proceeds. The third set contains the resolvents derived during the saturation process, none of which contain variable x.

The *MaxRes* process implements the MaxSAT resolution rule using either of the variants presented in Section 3.3.1. In practice, rather than removing and re-adding hard clauses or weighted clauses with weights greater than those of the other premise,

Algorithm 3.3: A generic algorithm for Weighted Partial MaxSAT saturation

Input: C, x: multiset of weighted and hard clauses containing variable x, and x.
Output: C': saturation of C with respect to x.

$C_x \leftarrow \{A | A \in C, x \in A\}$
$C_{\neg x} \leftarrow \{B | B \in C, \neg x \in B\}$
$C'_x \leftarrow C_x$; $C'_{\neg x} \leftarrow C_{\neg x}$; $L \leftarrow \{\}$
foreach $A \in C'_x$ **do**
  foreach $B \in C'_{\neg x}$ **do**
    if *not resolventIsTautology*(A,B) **then**
      $(A, B, Comp_x, Comp_{\neg x}, Res) \leftarrow \text{MaxRes}(A, B)$
      $C'_x \leftarrow C'_x \cup Comp_x$
      //Make sure that new clauses $Comp_x$ are visited in later iterations of the outer loop
      $C'_{\neg x} \leftarrow C'_{\neg x} \cup Comp_{\neg x}$
      //Make sure that new clauses $Comp_{\neg x}$ are visited in later iterations of the inner loop
      $L \leftarrow L \cup \{Res\}$
      if $weight(B) = 0$ **then**
        $C'_{\neg x} \leftarrow C'_{\neg x} \setminus \{B\}$
      if $weight(A) = 0$ **then**
        $C'_x \leftarrow C'_x \setminus \{A\}$
        break
$C' \leftarrow C \setminus C_x \setminus C_{\neg x} \cup C'_x \cup C'_{\neg x} \cup L$
return C'

we simply maintain the premise and decrease its weight as needed. Hence, the *MaxRes* process returns the premises A and B updated accordingly.

Theorem 3.4.7 states the correctness of Algorithm 3.3.

Lemma 3.4.6. *Let $x \vee A'$ be a compensation clause obtained after applying MaxSAT resolution with a soft premise $x \vee A$. Then, $A' \supseteq A$. This property is transitive: for any $A' \supseteq A$, the compensation clause $x \vee A''$ obtained with premise $x \vee A'$ also has the property $A'' \supseteq A$.*

Theorem 3.4.7. *Algorithm 3.3 generates a saturation of C w.r.t. x.*

Proof. It is easy to see that clauses in $C \setminus C_x \setminus C_{\neg x}$ do not contain variable x. In fact, they have been ignored during the process and have not been taken into account to obtain saturation. We need to show that at the end of the process:

- **For every pair of two hard clauses $A \in C'_x$, $B \in C'_{\neg x}$, their resolvent is in L (and hence in C').** Every such clause A has been visited in the outer loop, and since its weight never becomes 0 after applying *MaxRes*, it has been resolved against any hard clause in $C'_{\neg x}$ in the inner loop, including B. Note that *MaxRes* never adds hard compensation clauses during the whole process.

- **For every pair of two soft clauses $A \in C'_x$, $B \in C'_{\neg x}$, their resolvent is a tautology.** Every soft clause A remaining in C'_x has $weight(A) > 0$ and has been visited in the outer loop. Let us denote by $C'^{A}_{\neg x}$ the set of clauses $C'_{\neg x}$ right after the inner loop has finished in the iteration of the outer loop that considered clause A. Since $weight(A) > 0$, the resolvent of A and any clause in $C'^{A}_{\neg x}$ is a tautology (otherwise further MaxRes steps could be applied on A). Moreover, by Lemma 3.4.6, any new compensation clause added later to $C'_{\neg x}$ will contain all literals of some clause in $C'^{A}_{\neg x}$, hence generating a tautological resolvent if resolved against A. Hence, the property holds regardless of when B is added to $C'_{\neg x}$.

- **For every pair of hard clause $A \in C'_x$ and soft clause $B \in C'_{\neg x}$, their resolvent is a tautology.** Let us denote by $C'^{A}_{\neg x}$ the set of clauses $C'_{\neg x}$ right after the inner loop has finished in the iteration of the outer loop that considered clause A. Since the weight of a hard clause is ∞, the resolvent of A with any soft clause $B \in C'^{A}_{\neg x}$ is a tautology (otherwise clause B would have been removed when resolving against A). For the same reason as in the previous case, any new compensation clause added later to $C'_{\neg x}$ also yields a tautological resolvent when resolved against A.

- **For every pair of soft clause $A \in C'_x$ and hard clause $B \in C'_{\neg x}$, their resolvent is a tautology.** Hard clauses are never added nor removed from $C'_{\neg x}$ during the whole process. Hence, if clause A is still in C'_x at the end of the process, meaning $weight(A) > 0$, no hard clause in $C'_{\neg x}$ could be resolved against A without generating a tautology.

□

Notice the comments in Algorithm 3.3 that highlight the need to make sure that new compensation clauses are visited. Actually, the proof of Theorem 3.4.7 relies on this fact. In the following subsection, we present an implementation of this algorithm based on a FIFO (first-in-first-out) strategy to account for this. We also identify that certain properties of the management of compensation clauses introduce a critical threat to the performance of the process. Therefore, we next propose an alternative strategy, LIFO (last-in-first-out), which mitigates these threats. Finally, we provide more implementation details in the last subsection.

FIFO Variable Saturation

The idea of this strategy is quite simple: we should treat sets C'_x and $C'_{\neg x}$ on a FIFO basis. Hence, seeing these sets as lists initialized with some permutation, we iterate them in order from the beginning, and new clauses are added to the end. This ensures that all new clauses are visited in later iterations as required by Algorithm 3.3, unless the permitted case in which the algorithm leaves the inner loop with the break instruction when the weight of clause A has been exhausted.

This strategy, which is quite simple to implement, has important drawbacks arising from the following lemmas, which follow directly from the definition of the MaxSAT resolution rule.

Definition 3.4.8. *We say that two clauses of the form $x \vee A$, $\neg x \vee B$ are **incompatible** if $A \vee B$ is a tautology.*

Lemma 3.4.9. *Let $Comp_{\neg x}$ be the set of compensation clauses containing $\neg x$ generated from an application of MaxSAT resolution with a premise $x \vee A$. Then, every clause $\neg x \vee B \in Comp_{\neg x}$ is incompatible with $x \vee A$. The same property holds analogously between the set $Comp_x$ and its premise $\neg x \vee B$.*

Lemma 3.4.10. *Let $Comp_x$ and $Comp_{\neg x}$ be the sets of compensation clauses generated from an application of MaxSAT resolution. Then, every pair of clauses $x \vee A \in Comp_x$, $\neg x \vee B \in Comp_{\neg x}$ is incompatible.*

Consider, for instance, a formula with only two soft clauses of the same weight, resolvable on x, of the form $x \vee y_1 \vee y_2 \vee \cdots \vee y_n$ and $\neg x \vee z_1 \vee z_2 \vee \cdots \vee z_m$. By Lemma 3.4.10, this formula will become saturated with respect to x after a single application of MaxSAT resolution. As a result, $n+m$ compensation clauses will be generated. However, the FIFO strategy ignores that the formula has become immediately saturated, and will worthlessly check $n \times m$ pairs of incompatible clauses.

Having weighted clauses introduces further possible scenarios like having a clause $x \vee A$ coexisting in $C'_x \cup C'_{\neg x}$ at the same time with a set of compensation clauses $Comp_{\neg x}$ generated with $x \vee A$ as a premise. This would be the case if $Comp_{\neg x}$ is obtained after applying MaxSAT resolution between two clauses $(x \vee A, w_1)$ and $(\neg x \vee B, w_2)$, where $w_1 > w_2$. Again, now by Lemma 3.4.9, the FIFO strategy would waste time checking several incompatible pairs of clauses.

LIFO Variable Saturation

The variety of scenarios generating compensation clauses that will force the FIFO strategy to consider several incompatible pairs of clauses goes much further than the properties described in Lemmas 3.4.9 and 3.4.10. Most notably, the incompatibility between sibling positive and negative compensation clauses can be propagated transitively to new clauses obtained after using such compensation clauses as new premises in new MaxSAT resolution applications. The LIFO strategy is designed to capture this property, based on the following lemma, which subsumes and extends Lemmas 3.4.9 and 3.4.10.

Lemma 3.4.11. *Let $Comp_x$ and $Comp_{\neg x}$ be the sets of compensation clauses generated from an application of MaxSAT resolution on premises $x \vee A$ and $\neg x \vee B$. Then, it holds that any clause $x \vee A'$, where $A' \supseteq A$, is incompatible with every clause $\neg x \vee B \in Comp_{\neg x}$. Moreover, given a sequence of sets of positive compensation clauses $Comp_x^1, \ldots, Comp_x^n$, where the premise of $Comp_x^1$ is $x \vee A$, and for $i > 1$, the premise of $Comp_x^i$ is $E_{i-1} \in Comp_x^{i-1}$, we have that any $x \vee A' \in Comp_x^n$ satisfies $A' \supseteq A$.*

To take advantage of this property, when we pick a clause from C'_x, we will build a *compensation stack*. The purpose of this stack is to delay the addition of compensation clauses while we are considering premises that we know are incompatible. There is an entry in the stack for each generated positive compensation clause. Moreover, the stack satisfies that, given a sequence $Comp_x^1, \ldots, Comp_x^n$ of sets of compensation clauses like

the one defined in Lemma 3.4.11, for any two clauses $c_1 \in Comp_x^i$, $c_2 \in Comp_x^j$ with $i < j$, clause c_2 will be above c_1 in the compensation stack.

More precisely, an entry of the compensation stack is a pair (A,N), where A is a clause containing x and N is the set of negative compensation clauses generated with MaxRes applications with premise A. The addition to $C'_{\neg x}$ of clauses in N is delayed until all positive clauses in higher positions of the stack have been considered for saturation.

The following example illustrates the behavior of the LIFO strategy.

Example 3.4.12. *In this example, we will ignore the generated resolvents and only focus on compensation clauses. Consider the sets of weighted soft clauses*

$$C'_x = \{c_1{:}(x \vee y, 3)\}$$

$$C'_{\neg x} = \{c_2{:}(\neg x \vee z_1, 2), c_3{:}(\neg x \vee z_2, 1), c_4{:}(\neg x \vee z_3, 1)\}.$$

We first apply MaxRes(c_1, c_2). As a result, the weights of c_1 and c_2 are decreased by 2, c_2 has weight 0 and two compensation clauses $c_5{:}(x \vee y \vee \neg z_1, 2)$ and $c_6{:}(\neg x \vee \neg y \vee z_1, 2)$ should be added. However, we will delay adding these clauses. Instead, we will push the positive compensation clause c_5 onto the stack and record the negative clause c_6 in the list of negative compensation clauses for c_1. Now we apply MaxRes(c_1, c_3). The weights of these clauses are decreased to 0. There is one positive compensation clause $c_7{:}(x \vee y \vee \neg z_2, 1)$, and one negative clause $c_8{:}(\neg x \vee \neg y \vee z_2, 1)$. A new entry is pushed to the stack for c_7, and c_8 is added to the list of negative compensations of c_1. At this point, the stack contains the following:

TOP
$A = c_7{:}(x \vee y \vee \neg z_2, 1)$ $N =$
$A = c_5{:}(x \vee y \vee \neg z_1, 2)$ $N =$
$A = c_1{:}(x \vee y, 0)$ $N = c_6{:}(\neg x \vee \neg y \vee z_1, 2)$, $c_8{:}(\neg x \vee \neg y \vee z_2, 1)$

Now, we have $C'_{\neg x} = \{c_4{:}(\neg x \vee z_3, 1)\}$. Since the weight of c_1 is 0, we move to the top of the stack and consider the positive clause c_7. Note that c_6 and c_8, which are incompatible with c_7, are not considered as negative candidates. We pick the next negative candidate from $C'_{\neg x}$, which is c_4. MaxRes(c_7, c_4) will generate the positive compensation clause $c_9{:}(x \vee y \vee \neg z_2 \vee \neg z_3), 1)$, which is pushed to the stack, and associate to c_7 the negative compensation clauses $c_{10}{:}(\neg x \vee \neg y \vee z_3, 1)$ and $c_{11}{:}(\neg x \vee y \vee z_2 \vee z_3, 1)$. At this point, the stack contains the following:

TOP
$A = c_9{:}(x \vee y \vee \neg z_2 \vee \neg z_3), 1$ $N =$
$A = c_7{:}(x \vee y \vee \neg z_2, 0)$ $N = c_{10}{:}(\neg x \vee \neg y \vee z_3, 1), c_{11}{:}(\neg x \vee y \vee z_2 \vee z_3, 1)$
$A = c_5{:}(x \vee y \vee \neg z_1, 2)$ $N =$
$A = c_1{:}(x \vee y, 0)$ $N = c_6{:}(\neg x \vee \neg y \vee z_1, 2)$, $c_8{:}(\neg x \vee \neg y \vee z_2, 1)$

Since the weight of c_7 has become 0, we move c_7 to the top of the stack and consider the positive candidate c_9. None of the negative compensation clauses generated so far needs to be checked against c_9.

Now, we have $C'_{\neg x} = \{\}$; hence, c_9 is already saturated. It will be added to C'_x without being considered again, and it will be one of the clauses of the final saturated formula.

At this moment, we need to pop elements from the stack until we find the next entry with a clause A with non-zero weight. All negative compensation clauses from entries that we remove will be added to $C'_{\neg x}$. Such an entry is the one containing c_5, hence the content of the stack will be the following:

TOP
$A = c_5{:}(x \vee y \vee \neg z_1, 2)$ $N =$
$A = c_1{:}(x \vee y, 0)$ $N = c_6{:}(\neg x \vee \neg y \vee z_1, 2)$, $c_8{:}(\neg x \vee \neg y \vee z_2, 1)$

And we will have $C'_{\neg x} = \{c_{10}{:}(\neg x \vee \neg y \vee z_3 \vee, 1), c_{11}{:}(\neg x \vee y \vee z_2 \vee z_3 \vee, 1)\}$. The current positive candidate is c_5. The stack will still prevent to try to resolve it against c_6 or c_8, but now we can pick c_{11} from $C'_{\neg x}$, which until now was stored in an entry above.

This process continues until the stack is empty. Then, we will start a new stack with the next candidate in C'_x, if any.

An interesting additional property of the compensation stack is that it allows us to conclude that some positive conclusions are saturated without the need to visit them. This will happen whenever the positive premise that generated such a conclusion achieves saturation without exhausting its weight.

Example 3.4.13. *Consider the same example as before with two slight changes. First, the weight of clause c_1 is 5. Second, $C'_{\neg x}$ contains after c_4 some more clauses $c_{20}, c_{21}, \ldots$, all of them incompatible with c_1. Hence:*

$$C'_x = \{c_1{:}(x \vee y, 5)\}$$

$$C'_{\neg x} = \{c_2{:}(\neg x \vee z_1, 2), c_3{:}(\neg x \vee z_2, 1), c_4{:}(\neg x \vee z_3, 1), c_{20}, c_{21}, \ldots\}$$

If we follow the described process, we will be able to resolve c_2,c_3, and c_4 against c_1, and after this process, the weight of c_1 will be 1. The obtained stack will be:

TOP
$A = c_{12}{:}(x \vee y \vee \neg z_3, 1)$ $N =$
$A = c_7{:}(x \vee y \vee \neg z_2, 1)$ $N =$
$A = c_5{:}(x \vee y \vee \neg z_1, 2)$ $N =$
$A = c_1{:}(x \vee y, 1)$ $N = c_6{:}(\neg x \vee \neg y \vee z_1, 2)$, $c_8{:}(\neg x \vee \neg y \vee z_2, 1)$, $c_{13}{:}(\neg x \vee \neg y \vee z_3, 1)$

Note that c_1 has been checked against all candidate negative clauses and its weight is still 1. Hence, this clause is saturated. Any positive clause above c_1 in the stack is a superset of c_1, and hence it is saturated as well, so we can directly clear all elements above c_1.

Further Implementation Details

The order of the inner and outer loop in Algorithm 3.3 for the FIFO strategy is arbitrary; they could be interchanged, also interchanging the role of C'_x and $C'_{\neg x}$ elsewhere. Similarly, the roles of positive and negative compensation clauses in the compensation stack could also be interchanged in the LIFO strategy. Our implementation is designed to easily determine which polarity is dominant (the one of the outer loop or pushed into the stack) each time we start saturating a variable, without any overhead. At present, we always pick the positive literal, but heuristically selecting the dominant polarity for efficiency is an interesting future research line.

On the other hand, while applying MaxSAT resolution during saturation, self-subsuming resolution may occur between two hard clauses. When this happens, the subsumed premise (or both premises, if applicable) is immediately removed.

Finally, when adding new compensation clauses to the formula, one may ask whether they are already subsumed by a hard clause and can therefore be omitted. It is unclear whether considering all hard clauses in the formula, rather than only those involved in the saturation process, would improve the performance of MaxSAT saturation; we leave this for future work. At present, we check whether positive and negative compensation clauses are subsumed by any hard clause in $C'x$ and $C'\neg x$, respectively. This is done by forward subsumption. Our implementation takes into account the specific ladder structure of the compensation clauses generated by MaxSAT resolution. For instance, considering positive compensation clauses, recall from Definition 3.3.1 that they have the following form:

$$
\begin{aligned}
&c_1 : x \vee a_1 \vee \cdots \vee a_s \vee \neg b_1 \\
&c_2 : x \vee a_1 \vee \cdots \vee a_s \vee b_1 \vee \neg b_2 \\
&\cdots \\
&c_t : x \vee a_1 \vee \cdots \vee a_s \vee b_1 \vee \cdots \vee b_{t-1} \vee \neg b_t
\end{aligned}
$$

Once we determine that $x \vee a_1 \vee \cdots \vee a_s \vee \cdots \vee b_i$ is subsumed by some hard clause in C'_x, we can conclude that all clauses $c_{i+1} \ldots c_t$ will be subsumed as well. And, on the other way around, if we determine that no clause subsumes $x \vee a_1 \vee \cdots \vee a_s \vee b_i$, we

only need to check clauses containing $\neg b_{i+1}$ to determine if clause $x \vee a_1 \vee \cdots \vee a_s \vee \cdots \vee b_i \vee \neg b_{i+1}$ is subsumed.

3.4.5 Post-Saturation Process

So far, we have selected a variable x and saturated it. That is, starting from a set C, we have obtained C', which is a saturation of C with respect to x. Recall from Algorithm 3.3 that C' is defined as $C \setminus C_x \setminus C_{\neg x} \cup C'x \cup C'\neg x \cup L$. This saturation process is performed on copies of C_x and $C_{\neg x}$, without modifying the original formula, so that it can be aborted, if desired for BVE, without reverting the state of the formula. According to the general variable elimination process in Algorithm 3.2, the next step is to partition C' into two sets, C_i and D_i, where $C_i = C \setminus C_x \setminus C_{\neg x} \cup L$ and $D_i = C'x \cup C'\neg x$. The clauses in D_i play no further role until the model is reconstructed at the end of the solving process, and are therefore stored separately. In fact, MSElim stores only the smaller of $C'x$ and $C'\neg x$, since this is sufficient to reconstruct a model.

Now, we must remove from the current formula all clauses in C_x and $C_{\neg x}$, and add the clauses from L. As explained in Section 3.2, this process is well complemented with additional techniques such as clause subsumption, which we also perform. The full process is illustrated in Figure 3.4. During the process, we maintain separate lists to keep track of clauses that need to be backward or forward subsumed, clauses that need to be strengthened, and removed clauses. In Figure 3.4 these lists are named B, F, S and R, respectively. The Boolean attributes *removed*, *mark_strengthen*, and *mark_subsume* of the clauses (recall Section 3.4.2) will prevent having multiple references to the same clause in the same list.

We start the process by adding the new resolvents in L to the formula (step 1) and removing (storing aside) the clauses containing the eliminated variable (step 2). After this, we will apply various inference and simplification techniques until a fixpoint is reached. Note that we have enriched the Weighted Partial MaxSAT process with clause merging and Rules 1 and 2.

Unit propagation (B1). This must be run since set L, containing the conclusions of saturation, might include unit hard clauses. Every time we assign a literal ℓ, all clauses containing ℓ will be listed for removal, and clauses containing $\neg\ell$ will be listed for strengthening.

Clause strengthening (B2). We strengthen all listed clauses at the same time, i.e., we remove falsified literals from these clauses. These clauses become candidates to backward subsume.

Check clauses for forward and backward subsumption (B3). As usual in the literature, forward subsumption is implemented by inspecting the watch lists of the forward subsumed clause, and backward subsumption is implemented by inspecting the shortest occurrence list among the literals of the backward subsuming clause [18]. Moreover, during backward subsumption, when comparing the current clause c_1 with other clauses c_2, we will consider if any of the following mutually exclusive relations hold:

- There is a **subsumption** relation, hence one clause c_2 removed. Note that in the context of Weighted Partial MaxSAT, a clause c_1 will only subsume a clause c_2 if c_1 is hard, otherwise removing c_2 does not preserve the optimality.
- **Self-subsuming resolution** can be applied. The subsumed premises will be removed, and the new conclusion will be listed for backward subsumption.
- We can apply **clause merging**. If c_1 and c_2 are soft clauses that contain the same literals, they will be merged into a single clause by adding their weights. This operation preserves optimality. Note that if at least one of c_1 or c_2 is hard, this is handled as a subsumption relation.
- **Rule 1** can be applied. If c_1 and c_2 are soft clauses of the form $x \vee y$, $\neg x \vee y$, and there exists a unit soft clause $\neg y$, the three soft clauses can be replaced by an empty soft clause. This follows from two applications of MaxSAT resolution. With our representation of unit soft clauses, the existence of clause $\neg y$ can be checked in constant time.

The abstractions of c_1 and c_2 are used to check if the previous relations between the clauses hold. This check can produce false positives, but it allows us to avoid a significant number of traversals over the literals of the clauses. Finally, if c_1 still exists after being checked for backward subsumption and is of the form $x \vee y$, we will apply **Rule 2** if possible. This rule specifies that from three clauses of the form $x \vee y$, $\neg x, \neg y$, where at least the latter two are soft, we can derive an empty clause and clause $\neg x \vee \neg y$, and remove the soft premises. Again, this comes from two applications of MaxSAT resolution. Rules 1 and 2 have already been used before in the MaxSatz solver [68].

The application of self-subsuming resolution might have added new unit hard clauses; if so, we will iterate the whole process again from unit propagation.

All the changes we made to the formula up to step B3 can affect the scores of several variables in the variable selection heuristics. To avoid recomputing the scores of all variables in the formula, which would be very time consuming, we will identify the variables that might have changed their scores. These are the variables that appear in new, deleted, or strengthened clauses. This set of clauses can be retrieved from the lists B and R (step 3), and the scores are finally recomputed (step 4) as explained in Section 3.4.3.

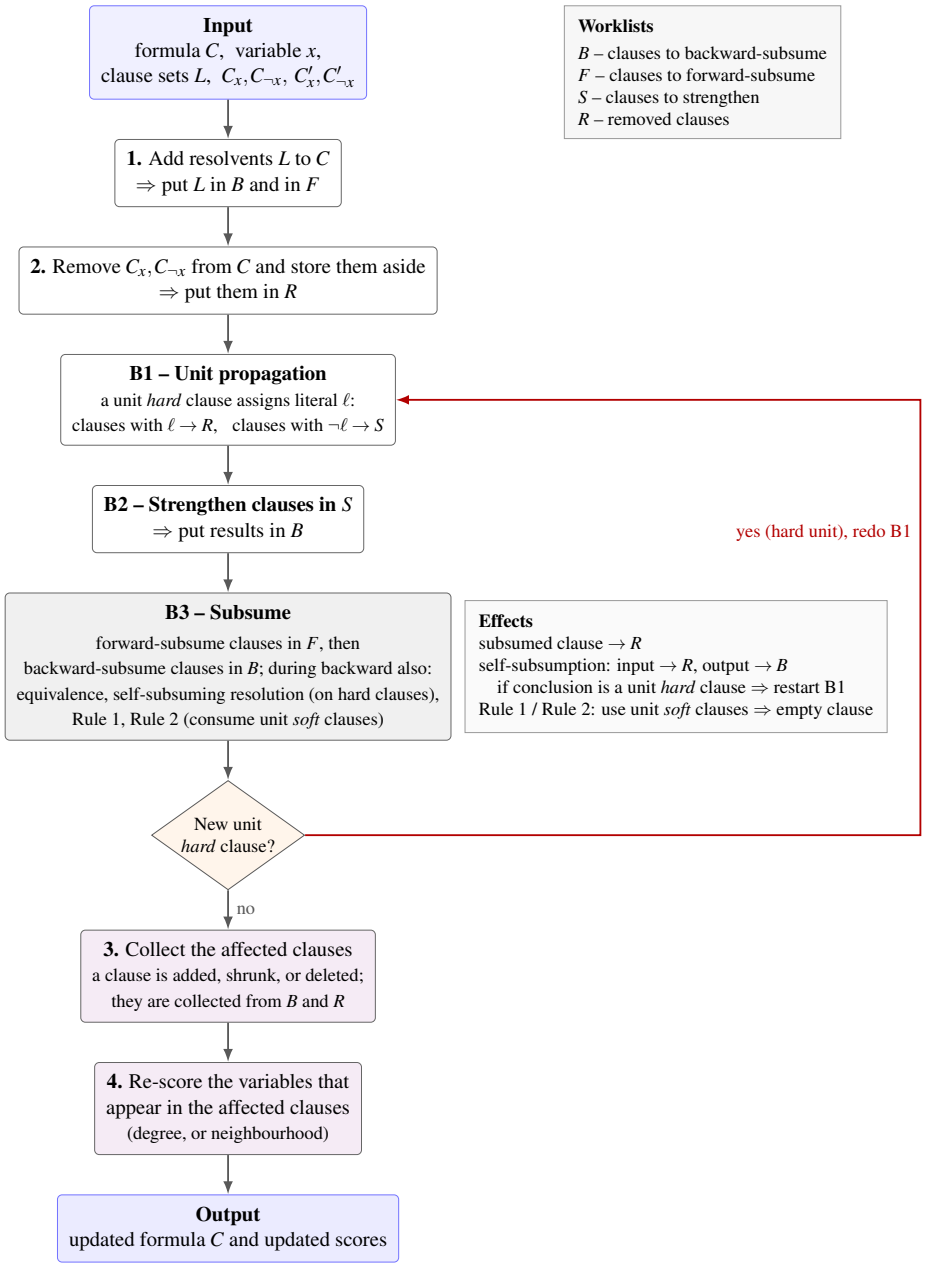


Figure 3.4: Post-saturation process for eliminating a variable in MSElim.

3.5 Experimental Evaluation

This section presents an experimental evaluation of MSElim. Subsection 3.5.1 evaluates MSElim as a complete solver by comparing different configurations. Subsection 3.5.2 presents initial results demonstrating the potential of MSElim as a BVE preprocessor for MaxSAT.

3.5.1 MSElim as a Solver

To evaluate the performance of MSElim, we solved instances from the 2022, 2023 and 2024 editions of the MaxSAT Evaluation. Specifically, we focused on the instances from the exact weighted and unweighted tracks. After removing duplicates, the dataset comprised 1.558 unweighted instances and 1.545 weighted instances. The experiments were executed on an AMD EPYC multiprocessor model 9755 TURIN at 2.7 GHz. Each execution was limited to 16 GB of memory and a time cutoff of 1 hour.

The results are reported in Figures 3.5 and 3.6, corresponding to the unweighted and weighted sets, respectively. As expected, these results are not competitive with state-of-the-art MaxSAT solvers (not reported here), which, for example, solved more than 400 of the 571 instances in the exact weighted track of MaxSAT Evaluation 2024 [14, 15]. In fact, the final goal of MSElim is not to be a competitive solver, but rather a powerful preprocessor. Accordingly, the purpose of our first results is to show that the techniques introduced are critical to speed up the variable elimination process. We analyze the effect of the following five parameters:

heur: Variable selection heuristic. The possible values are 0, 1, 2, and 3, coinciding with the numeration presented in Section 3.4.3.

stra: Strategy used to saturate a variable (Section 3.4.4). The possible values are 0 (FIFO) and 1 (LIFO).

sub: Degree of subsumption (Section 3.4.5). A value of 0 disables subsumption, 1 enables backward subsumption, and 2 enables both backward and forward subsumption.

elim-sub: If true (T), compensation clauses are checked for subsumption during the saturation process (see the last subsection of Section 3.4.4).

s-r12: If true (T), during backward subsumption, we also try to apply self-subsuming resolution and Rules 1 and 2.

The first result to highlight is the large performance difference between a naive implementation (heur=0, stra=0, sub=0, elim-sub=F and s-r12=F) and the best-performing configuration (heur=2, stra=1, sub=2, elim-sub=T and s-r12=T). The former solves only 36 and 49 instances from the unweighted and weighted sets, respectively, whereas the latter solves 162 and 136 instances, respectively.

In the following subsections, we perform an ablation study by varying one parameter at a time while keeping the remaining parameters fixed to the values of the best-performing configuration, namely heur=2, stra=1, sub=2, elim-sub=T and s-r12=T.

Comparison between Heuristics

To assess the impact of the variable selection heuristic, we compare the four heuristics presented in Section 3.4.3, keeping the remaining parameters fixed. The relative ranking of the heuristics is the same for both sets. Heuristic 2 achieves the best results, solving 162 unweighted and 136 weighted instances, followed by Heuristic 1 with 146 and 116 instances, and Heuristic 3 with 137 and 102. The lexicographic baseline, Heuristic 0, solves 75 and 73 instances, fewer than half the number solved by Heuristic 2. This confirms that the order in which variables are eliminated is a major performance factor.

Interestingly, Heuristic 3 is outperformed by Heuristic 2, which only approximates the neighbourhood size. Although Heuristic 3 computes the exact neighbourhood size, its score computation requires traversing every literal of every clause containing the variable, and the improvement in accuracy does not compensate for this additional cost.

Comparison between FIFO and LIFO

To compare the two saturation strategies presented in Section 3.4.4, we consider two configurations that differ only in the saturation strategy. LIFO solves 162 instances from the unweighted set and 136 from the weighted set, whereas FIFO solved 151 and 126, respectively. This is consistent with the design of the LIFO strategy, which avoids checking multiple incompatible pairs of compensation clauses that the FIFO strategy considers. We also assess the impact of checking compensation clauses for subsumption during the saturation process. Disabling this check in the LIFO configuration reduces the number of solved instances to 157 and 134, suggesting that discarding subsumed compensation clauses helps contain the growth of the formula.

Impact of Subsumption Techniques

Finally, we assess the impact of the subsumption techniques presented in Section 3.4.5. Disabling subsumption completely reduces the number of solved instances to 66 in both sets, demonstrating that subsumption is essential for the performance of the elimination process. Enabling backward subsumption alone increases this number to 140 in the unweighted set and 120 in the weighted set, and adding forward subsumption further increases it to 157 and 126. Regarding self-subsuming resolution and Rules 1 and 2, enabling them on top of full subsumption increases the number of solved instances from 157 to 162 in the unweighted set and from 126 to 136 in the weighted set.

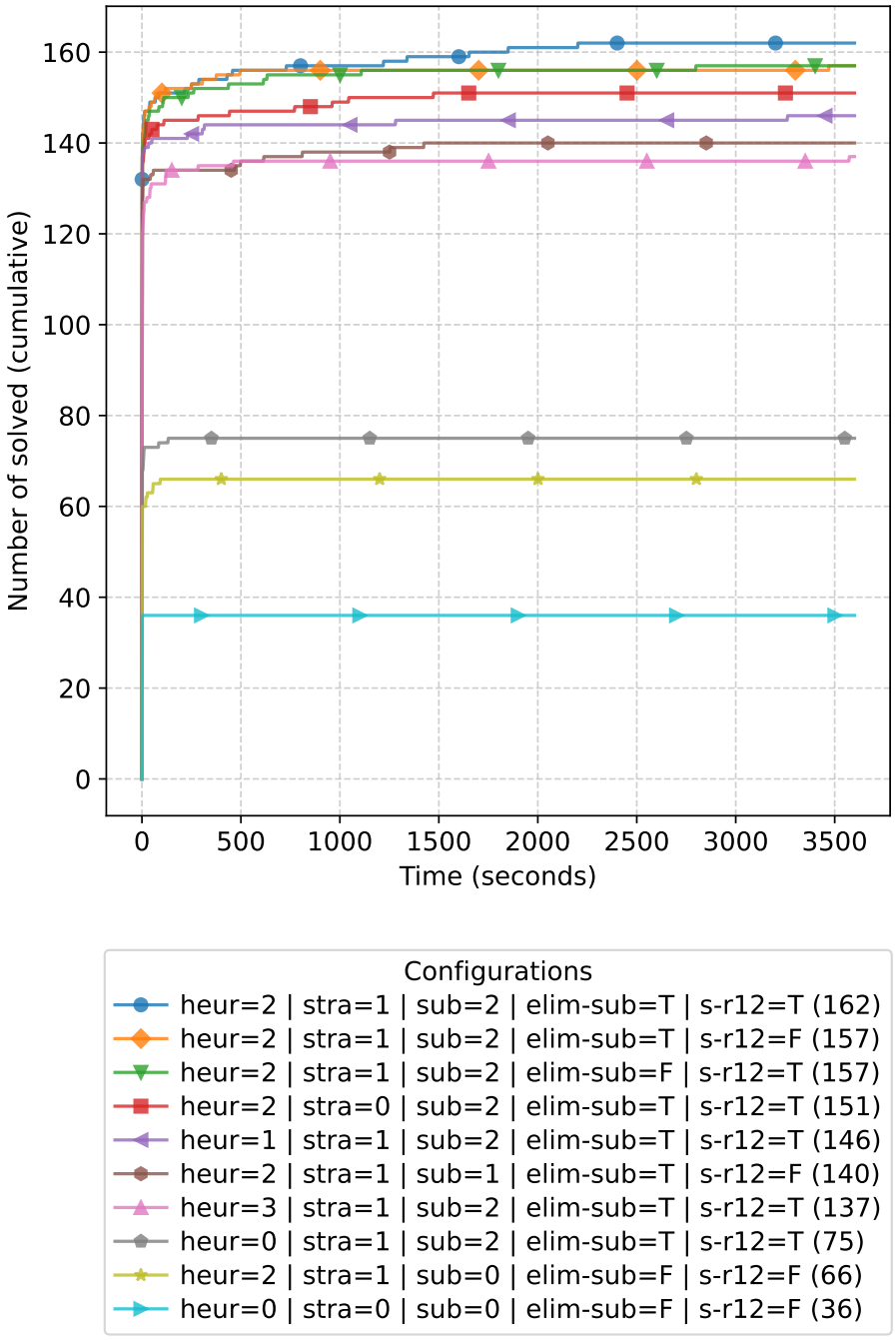


Figure 3.5: Cactus plot of the different configurations for solving the unweighted set. The number of solved instances is shown in the legend.

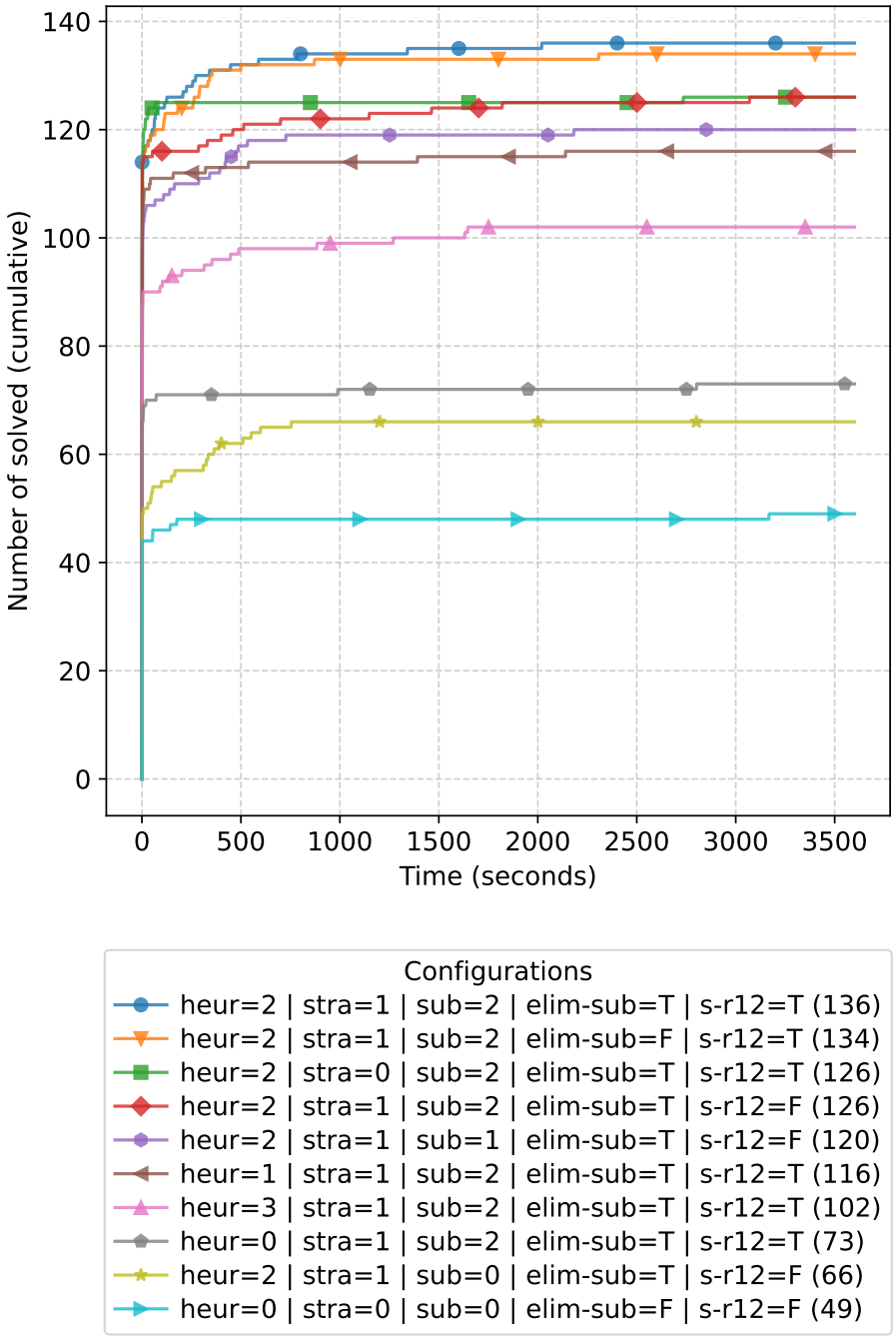


Figure 3.6: Cactus plot of the different configurations for solving the weighted set. The number of solved instances is shown in the legend.

Dataset	CGSS2			Pacose			WMaxCDCL		
	#s	#e	#f	#s	#e	#f	#s	#e	#f
MSE22-u	390	21	194	303	42	138	397	5	181
Pre-MSE22-u	389	20	175	275	14	123	389	13	203
MSE22-w	407	29	155	330	47	131	413	33	222
Pre-MSE22-w	384	6	222	297	14	152	385	5	158

Table 3.3: Results solving unweighted (`u`) and weighted `w` instances of the exact track of MaxSAT Evaluation 2022. `Pre` denotes that MSElim was used. For each solver and configuration, we show: `#s`: number of *solved* instances; `#e`: number of instances that were solved *exclusively* without/with using the preprocessing with that solver in that set; `#f`: number of instances that were solved both without/with using the preprocessing and were solved *faster* with that solver in that set.

3.5.2 MSElim as a BVE Preprocessor

We report initial results on using MSElim as a BVE preprocessor to generate instances that are then solved by state-of-the-art MaxSAT solvers. To assess its effectiveness, we use three solvers from MaxSAT Evaluation 2024 [14]: CGSS2, Pacose, and WMaxCDCL. WMaxCDCL has a counterpart solver, MaxCDCL, specialized for unweighted instances. However, we also use WMaxCDCL to solve unweighted instances because, after preprocessing, many originally unweighted instances become weighted due to clause merging. We selected these solvers for two reasons. First, they represent the three main MaxSAT solving approaches participating in the last editions of the MaxSAT Evaluation: core-guided solving, model-based solving, and branch and bound with clause learning. Second, none of them is a sequential composition of different solving techniques. In recent editions of the MaxSAT Evaluation, all top-performing solvers have sequentially divided the one-hour time limit among completely different solving techniques. Although this approach is effective for solving a broader range of instance families and achieving strong overall results, it makes it difficult to determine which MaxSAT solving algorithm benefits the most from MSElim's preprocessing.

We used the preprocessing configuration that reported the best results in the previous section: heur=2, stra=1, sub=2, elim-sub=T, s-r12=T. We followed the policy used by the implementation of NIVER for SAT [91], which decides that a variable x should not be eliminated if the sum of lengths of the clauses of the formula would increase after eliminating that variable. This is implemented by checking the conclusions obtained during the saturation step, and aborting the process if the sum of sizes of the conclusions (that will be the added clauses) reaches the sum of sizes of the clauses containing x or $\neg x$ when the process starts (that will be the removed clauses). On the other hand, we observed that once we aborted the elimination of some variables, the next selected variables (which have higher scores) are likely to be omitted. Moreover, the process of determining that a variable should not be eliminated becomes slower as the scores increase. Hence, to reduce preprocessing time, we stop eliminating variables after skipping 100 consecutive variables.

The total time limit, including preprocessing and solving, is one hour. These experiments were run on a cluster of Intel Xeon Silver 4210 CPUs at 2.2 GHz, with an

8 GB memory limit per task.

We preprocessed all instances from the exact tracks of the 2022 edition of the MaxSAT Evaluation, namely 594 unweighted instances and 607 weighted instances. The results are shown in Table 3.3. For Pre-MSE22-u and Pre-MSE22-w, among the solved instances, 7 and 8, respectively, were solved completely during preprocessing, despite the constraints imposed on BVE. On the other hand, 4 and 23 of the unsolved instances, respectively, reached the one-hour cutoff or ran out of memory during preprocessing.

The first conclusion from Table 3.3 is that preprocessing generally worsens solving performance, except for CGSS2 for MSE22-u. However, a deeper analysis of columns `#e` and `#f` suggests that the results are promising. The scatter plots in Figure 3.7 provide further insight into this per-instance comparison. Although fewer instances are solved when preprocessing is enabled, there is no general slowdown in solving times. On the contrary, a significant number of instances are solved faster with preprocessing, and many of them exhibit substantial speedups.

The most notable result is that CGSS2 solves 20 additional unweighted instances thanks to preprocessing. We inspected the instances that benefit from preprocessing and observed that many belong to particular benchmark families for which almost no instance is solved faster without MSElim. Therefore, these speedups are unlikely to be caused by the randomness introduced by changing the formula. Instead, they suggest that MSElim is particularly well suited to certain classes of instances. Below, we list some of the benchmark families that benefit most clearly from preprocessing, identified by the prefix of the instance name:

GGGS2 unweigthed: causal, extension-enforcement, RSN_Security_Min_Witness.

CGSS2 weighted: drmx-am, metro, power-distribution.

Pacose unweighted: mul.role, vio.role.

Pacose weighted: eas, limits, metro, test.

WMaxCDCL unweighted: extension-enforcement, navigation, vio.role.

WMaxCDCL weighted: Bounded_BTWBNSL, eas, test.

Another interesting observation is that each solver mainly benefits from preprocessing on different instance families, suggesting that different preprocessing configurations should be explored for different MaxSAT solving approaches.

3.6 Conclusions

We have presented our Weighted Partial MaxSAT solver and BVE preprocessor MSElim. After identifying the main performance bottlenecks in MaxSAT-based saturation resolution, we have introduced techniques and implementation details that deliver significant performance improvements. Moreover, we have presented results on BVE preprocessing that clearly suggest this approach has the potential to become a de facto standard preprocessing technique for MaxSAT solvers.

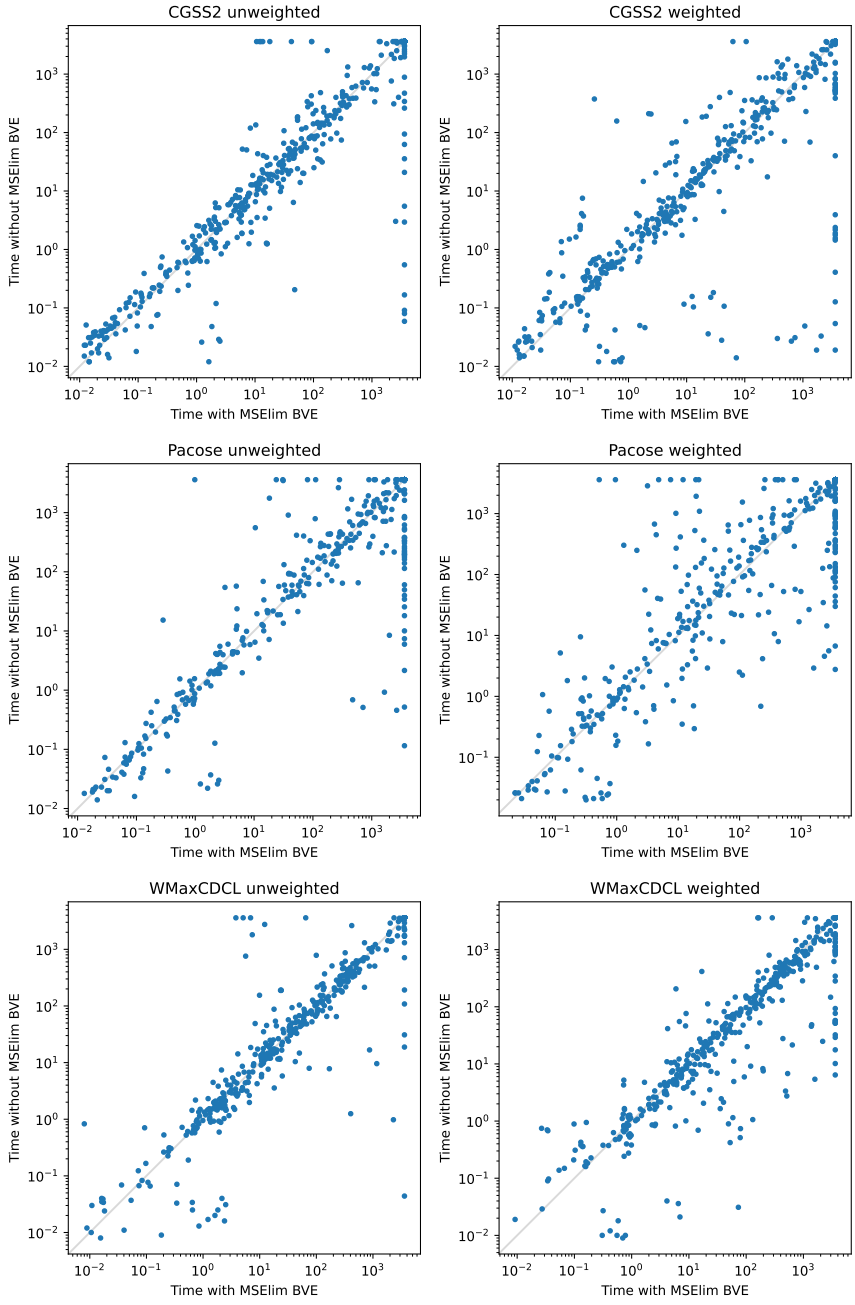


Figure 3.7: Scatter plots comparing the solving times of each instance using and without using MSElim's preprocessing.

Chapter 4

A Resolution Calculus for Regular MaxSAT

This chapter introduces a novel MaxSAT resolution calculus for regular propositional logic and establishes its soundness and completeness. It also describes a variable elimination algorithm that implements the rules of the defined calculus, and extends the calculus to Weighted Regular MaxSAT.

4.1 Introduction

The Boolean MaxSAT resolution calculus was extended to signed CNF formulas in [2, 5] to provide a formal framework for reasoning with soft constraints. That work introduced the simplest variant of signed logic, in which a sign S is a subset of a given truth value set N, a signed literal is an expression of the form $S:x$, where x is a propositional variable and S is a sign, a signed clause is a disjunction of signed literals, and a signed CNF formula is a multiset of signed clauses. Since the sign is attached to each literal rather than to the clause as a whole, a single clause may contain multiple distinct signs.

Within this framework, an assignment v maps propositional variables to truth values in N. The assignment v satisfies a signed literal $S:x$ if and only if $v(x) \in S$, satisfies a signed clause if it satisfies at least one of its literals, and satisfies a signed CNF formula if it satisfies all its clauses. Consequently, the semantics remain Boolean at the clause and formula levels. A sign is called *regular* when it is restricted to subsets of the form $\{j \in N \mid j \geq i\}$, denoted by $\geq i$, or $\{j \in N \mid j \leq i\}$, denoted by $\leq i$, for some $i \in N$. A clause is regular if all its signs are regular, and a regular CNF formula is a multiset of regular clauses.

This chapter considers precisely this regular-clause formalism. In contrast, the remainder of the thesis adopts the more general signed logic introduced in Chapter 2, where each formula A is associated with a single sign S. A valuation v assigns a truth value in N to every formula, and satisfies $S:A$ precisely when $v(A) \in S$. In that frame-

work, conjunction, disjunction, and negation are interpreted by the minimum, maximum, and Łukasiewicz negation operators, respectively.

This chapter introduces a novel Regular MaxSAT resolution calculus for regular CNF formulas. Since the resolution rule operates exclusively on clausal form, the input instances are multisets of regular clauses. To the best of our knowledge, this is the first MaxSAT resolution calculus specifically designed for this class of formulas.

Developing a Regular MaxSAT resolution calculus is motivated by several considerations. First, the structure of regular signs enables a simpler and more efficient variable elimination algorithm: testing whether two signs intersect requires only basic numerical comparisons, and each regular sign can be represented compactly by its boundary value. Second, the polarity inherent in regular signs allows the resolution rules to detect contradictions using only two literals, whereas general signed logic may require several literals. Third, regular CNF formulas constitute the underlying language of the order encoding [92], one of the most effective encodings for modeling cardinality constraints. Finally, regular signs naturally accommodate both finite and infinite truth value sets.

The remainder of this chapter is organized as follows. Section 4.2 introduces the Regular MaxSAT resolution calculus and proves its soundness and completeness. Section 4.3 presents an exact variable elimination algorithm based on the calculus. Section 4.4 extends the framework to Weighted Regular MaxSAT. Finally, Section 4.5 concludes the chapter.

4.2 Regular MaxSAT Resolution

This section defines a resolution calculus for Regular MaxSAT and proves its soundness and completeness. The calculus starts with a multiset of regular clauses C, and applies a set of inference rules that remove two clauses (premises) from C and introduce a number of new clauses (conclusions) to C.

When dealing with finitely-valued logics, the truth value set is usually taken to be $N = \{0, \frac{1}{n-1}, \ldots, \frac{n-1}{n-1}\}$, that is, the set of the n equidistant rational values in the unit interval. For the purposes of this chapter, we use the equivalent representation $N = \{1, \ldots, |N|\}$ to follow the notation of [2, 5] and maintain consistency with the terminology of the constraint programming community. All the results presented remain valid for the standard truth value set over the unit interval and require only minor adjustments.

We require all clauses to be in normalized form before any rule is applied. Assuming that $N = \{1, \ldots, |N|\}$, a clause can be normalized by applying the following simple transformations upon the first introduction of a clause or when it is derived:

- Tautologies are removed from C. A regular clause is a tautology if it contains either a regular literal of the form $\geq 1 : x$ or $\leq |N| : x$, or two regular literals $\leq i : x$ and $\geq j : x$ such that $j \leq i + 1$.
- If a clause contains multiple regular literals with positive polarity for a variable x, say $\{\geq i_1\} : x, \ldots, \{\geq i_k\} : x$, they are replaced with $\bigcup_{j=1}^{k} \{\geq i_j\} : x$, which also is a regular literal with positive polarity. Similarly, if the clause contains multiple

regular literals with negative polarity for variable x, say $\{\leq i_1\}:x,\ldots,\{\leq i_{k'}\}:x$, they are replaced with $\bigcup_{j=1}^{k'}\{\leq i_j\}:x$, which is also a regular literal with negative polarity.

These transformations are sound since they do not alter the set of satisfying assignments of a clause, and can be applied in $O(nlog(n))$ time in the size of the clause. The following inference rules assume that the premises are normalized and require the same normalization to be enforced on the conclusions.

Definition 4.2.1. *The following inference rules define the Regular MaxSAT resolution calculus:*

Rule 1

$$\begin{array}{lll} c_\alpha & : & \geq j:x\vee A \\ c_\beta & : & \leq k:x\vee B \\ \hline c_\cap & : & A\vee B \\ c_\cup & : & \leq k:x\vee\geq j:x\vee A\vee B \\ C_\alpha & : & \geq j:x\vee A\vee\neg B \\ C_\beta & : & \leq k:x\vee\neg A\vee B \end{array}$$

provided that $k<j$ and $A\vee B$ is not a tautology

Rule 2

$$\begin{array}{lll} c_\alpha & : & \geq j:x\vee A \\ c_\beta & : & \leq k:x\vee\geq l:x\vee B \\ \hline c_\cap & : & \geq l:x\vee A\vee B \\ c_\cup & : & \leq k:x\vee\geq j:x\vee A\vee B \\ C_\alpha & : & \geq j:x\vee A\vee\neg B \\ C_\beta & : & \leq k:x\vee\geq l:x\vee\neg A\vee B \end{array}$$

provided that $k<j<l$ and $A\vee B$ is not a tautology

Rule 3

$$\begin{array}{lll} c_\alpha & : & \leq i:x\vee\geq j:x\vee A \\ c_\beta & : & \leq k:x\vee B \\ \hline c_\cap & : & \leq i:x\vee A\vee B \\ c_\cup & : & \leq k:x\vee\geq j:x\vee A\vee B \\ C_\alpha & : & \leq i:x\vee\geq j:x\vee A\vee\neg B \\ C_\beta & : & \leq k:x\vee\neg A\vee B \end{array}$$

provided that $i<k<j$ and $A\vee B$ is not a tautology

Rule 4

$$\begin{array}{lll} c_\alpha & : & \leq i:x\vee\geq j:x\vee A \\ c_\beta & : & \leq k:x\vee\geq l:x\vee B \\ \hline c_\cap & : & \leq i:x\vee\geq l:x\vee A\vee B \\ c_\cup & : & \leq k:x\vee\geq j:x\vee A\vee B \\ C_\alpha & : & \leq i:x\vee\geq j:x\vee A\vee\neg B \\ C_\beta & : & \leq k:x\vee\geq l:x\vee\neg A\vee B \end{array}$$

provided that $i<k<j<l$ and $A\vee B$ is not a tautology

where $A=a_1\vee\cdots\vee a_s$ and $B=b_1\vee\cdots\vee b_t$ are disjunctions of regular literals not containing variable x. The rules replace the premises with the clauses in the conclusion, and we say that they resolve the premises on the variable x. The tautologies concluded by the rule are omitted. Note that the literals of the last two conclusions are a superset of the literals of one of the premises.

Note that $\neg A$ and $\neg B$ are not in clausal form in the rules. To build clausal forms preserving the number of unsatisfied clauses, we use $\neg A=\{\neg a_1, a_1\vee\neg a_2,\cdots,a_1\vee\cdots\vee a_{s-1}\vee\neg a_s\}$ and $\neg B=\{\neg b_1, b_1\vee\neg b_2,\cdots,b_1\vee\cdots\vee b_{t-1}\vee\neg b_t\}$. Therefore, C_α and C_β are sets of clauses. Figure 4.1 displays Rule 1 in clausal form.

Figure 4.2 illustrates the application of Rules 1–4 to pairs of clauses that only contain variable x. Note that we establish an order between c_α and c_β in Rule 4 to avoid defining symmetric cases. Specifically, c_α is defined as the clause with the smallest

$$
\begin{array}{ll}
c_\alpha : & \geq j : x \vee a_1 \vee \cdots \vee a_s \\
c_\beta : & \leq k : x \vee b_1 \vee \cdots \vee b_t \\
\hline
c_\cap : & a_1 \vee \cdots \vee a_s \vee b_1 \vee \cdots \vee b_t \\
c_\cup : & \geq j : x \vee \leq k : x \vee a_1 \vee \cdots \vee a_s \vee b_1 \vee \cdots \vee b_t \\
C_{\alpha,1} : & \geq j : x \vee a_1 \vee \cdots \vee a_s \vee \neg b_1 \\
C_{\alpha,2} : & \geq j : x \vee a_1 \vee \cdots \vee a_s \vee b_1 \vee \neg b_2 \\
\cdots & \\
C_{\alpha,t} : & \geq j : x \vee a_1 \vee \cdots \vee a_s \vee b_1 \vee \cdots \vee b_{t-1} \vee \neg b_t \\
C_{\beta,1} : & \leq k : x \vee b_1 \vee \cdots \vee b_t \vee \neg a_1 \\
C_{\beta,2} : & \leq k : x \vee b_1 \vee \cdots \vee b_t \vee a_1 \vee \neg a_2 \\
\cdots & \\
C_{\beta,s} : & \leq k : x \vee b_1 \vee \cdots \vee b_t \vee a_1 \vee \cdots \vee a_{s-1} \vee \neg a_s
\end{array}
$$

Figure 4.1: Rule 1 of the Resolution Regular MaxSAT calculus in clausal form.

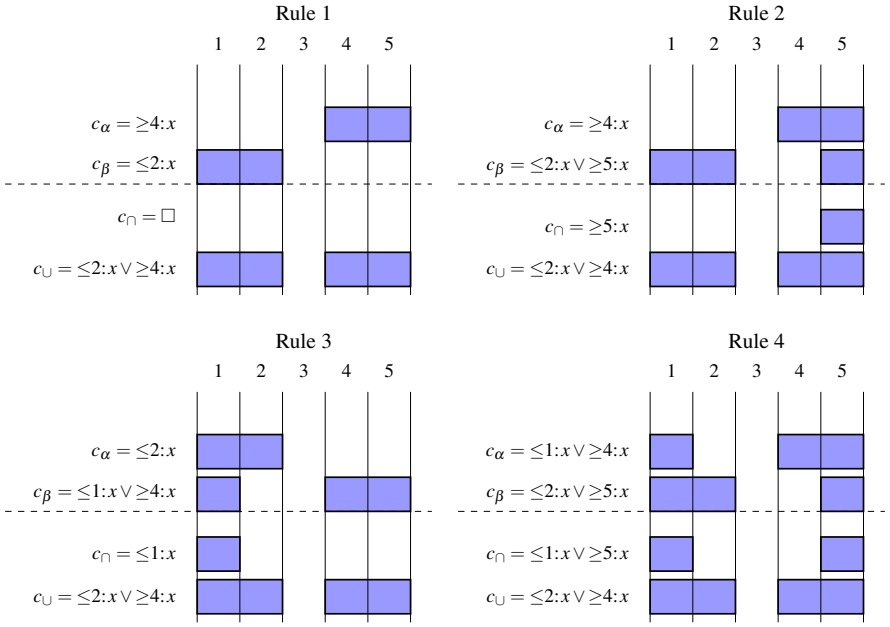


Figure 4.2: Examples of rule applications in the Resolution Regular MaxSAT calculus to clauses only containing variable x, with $N = \{1, \ldots, 5\}$. The values of x that satisfy each clause are highlighted in blue.

negative literal. Similarly, from now on, when considering a pair of clauses c_α and c_β that both contain positive and negative literals for the same variable x, we always assume that c_α has the smallest negative literal. In case of a tie, c_α has the smallest (or equal) positive literal.

Example 4.2.2. *If $N = \{1,2,3,4,5\}$, Rule 1 derives the multiset of regular clauses $\{\square, \leq 3 : x_1 \vee \geq 5 : x_1\}$ from the multiset $\{\leq 3 : x_1, \geq 5 : x_1\}$. Rule 2 derives the multiset of regular clauses $\{\geq 5 : x_1 \vee \geq 4 : x_2, \leq 2 : x_1 \vee \geq 4 : x_1 \vee \geq 4 : x_2\}$ from the multiset $\{\geq 4 : x_1 \vee \geq 4 : x_2, \leq 2 : x_1 \vee \geq 5 : x_1 \vee \geq 4 : x_2\}$. Rule 4 derives the multiset of regular clauses $\{\leq 1 : x_1 \vee \geq 5 : x_1 \vee \leq 2 : x_2, \leq 2 : x_1 \vee \geq 4 : x_1 \vee \leq 2 : x_2, \leq 2 : x_1 \vee \geq 5 : x_1 \vee \geq 3 : x_2\}$ from the multiset $\{\leq 1 : x_1 \vee \geq 4 : x_1 \vee \leq 2 : x_2, \leq 2 : x_1 \vee \geq 5 : x_1\}$.*

Theorem 4.2.3. *The Regular MaxSAT resolution calculus is sound.*

Proof. Given an assignment v, we prove that the number of unsatisfied clauses in the premises is the same as the number of unsatisfied clauses in the conclusion of the rule. We distinguish the following cases:

1. v falsifies the two premises: Since $v(A) = v(B) = v(\leq i : x) = v(\geq j : x) = v(\leq k : x) = v(\geq l : x) = false$, v falsifies the first two conclusions. It satisfies the other conclusions because either A or B appears negated in them. This holds for the four rules.

2. v satisfies the two premises: this case holds since the conjunction of the premises implies each one of the conclusions. More precisely, for all the rules, v satisfies the last two conclusions because their literals are a superset of the literals of some premise. This also holds for the second conclusion of Rules 1–3. For the second conclusion of Rule 4 we have three cases: (i) v satisfies A or B: v satisfies the second conclusion because it contains A and B; (ii) v falsifies A and B and satisfies $\geq j : x$: v satisfies the second conclusion because it contains $\geq j : x$; and (iii) v falsifies A and B and satisfies $\leq i : x$: Since $i < l$, v falsifies $\geq l : x$ and, since the second premise is satisfied, v satisfies $\leq k : x$. We now prove separately that v satisfies the first conclusion for each rule. For Rule 1, since $k < j$, v must satisfy A or B and, therefore, it satisfies the first conclusion. For Rule 2, we distinguish two cases: (i) v satisfies $\geq l : x$: It satisfies the first conclusion because it contains $\geq l : x$; and (ii) v falsifies $\geq l : x$: since $k < j$, v must satisfy A or B and, therefore, it satisfies the first conclusion. For Rule 3, we distinguish two cases: (i) v satisfies $\leq i : x$: It satisfies the first conclusion because it contains $\leq i : x$; and (ii) v falsifies $\leq i : x$: since $k < j$, v must satisfy A or B and, therefore, it satisfies the first conclusion. For Rule 4, we distinguish two cases: (i) v satisfies $\leq i : x$ or $\geq l : x$: It satisfies the first conclusion because it contains both literals; and (ii) v falsifies $\leq i : x$ and $\geq l : x$: since $k < j$, v must satisfy A or B and, therefore, it satisfies the first conclusion.

3. v satisfies the first premise and falsifies the second premise: For Rules 1–2, the second and third conclusions are implied by the first premise. Then, we distinguish two cases: (i) v satisfies A: the first conclusion is satisfied because it contains A and the last conclusion is unsatisfied because we assumed that all its literals are unsatisfied; and (ii) v falsifies A: the first conclusion is unsatisfied because we assumed that all its literals are unsatisfied and the last conclusion is satisfied because it contains $\neg A$. For Rules 3–4, we distinguish two cases: (i) v satisfies A: the three first conclusions are satisfied because they contain A and the last conclusion is clearly unsatisfied; and (ii) v falsifies A: $\leq i : x$ is unsatisfied

because $\leq k:x$ is unsatisfied and $i < k$ and, therefore, $\geq j:x$ is satisfied. Thus, the first conclusion is unsatisfied and the rest are satisfied.

4. v falsifies the first premise and satisfies the second premise. For Rule 1 and Rule 3, the second and fourth conclusions are implied by the second premise. Then, we distinguish two cases: (i) v satisfies B: the first conclusion is satisfied because it contains B and the third conclusion is unsatisfied because we assumed that all its literals are unsatisfied; and (ii) v falsifies B: the first conclusion is unsatisfied because we assumed that all its literals are unsatisfied and the third conclusion is satisfied because it contains $\neg B$. For Rule 2 and Rule 4, we distinguish two cases: (i) v satisfies B: the first, second and fourth conclusions are satisfied because they contain B and the third conclusion is clearly unsatisfied; and (ii) v falsifies B: $\geq l:x$ is unsatisfied because $\geq j:x$ is unsatisfied and $j < l$ and, therefore, $\leq k:x$ is satisfied. Thus, the first conclusion is unsatisfied and the rest are satisfied.

□

Definition 4.2.4. *A multiset C of regular clauses is saturated w.r.t. a variable x if, for every pair of regular clauses c_α and c_β containing x, none of the rules of the calculus (Rule 1, Rule 2, Rule 3 and Rule 4) can be applied.*

Figure 4.3 shows an example of a saturated multiset of regular clauses w.r.t. variable x.

Lemma 4.2.5. *Let $C = E \cup D$ be a saturated multiset of regular clauses w.r.t. a variable x, let E be the subset of clauses of C not containing x, and let D be the subset of clauses of C containing x. Assume that E is satisfiable. Then, any partial assignment v that satisfies E and does not assign any value to x can be extended to an assignment that satisfies C.*

Proof. We have to extend v so that v satisfies C. We partition D into the multiset D' and D'', where D' is not satisfied by v and D'' is satisfied by v. D' contains clauses of one of the following forms: $\leq i:x \vee \geq j:x \vee A$, $\leq i:x \vee A$, or $\geq j:x \vee A$. Since A is falsified in D', we assume that D' contains clauses without A in the remainder of the proof. Now, we define the set of values that x can take according to v, and show that it is never empty. If D' is empty, v can assign to x any value in N. From the set of unit clauses whose literal has negative polarity (i.e, of the form $\leq i:x$), we only need to consider the clause with the smallest i because the other negative literals are subsumed. Similarly, from the set of unit clauses whose literal has positive polarity (i.e., of the form $\geq j:x$), we only need to consider the clause with the greatest j. Therefore, the set of values that x can take is contained in the interval $[a,b]$, where the most restrictive unit clauses are $\geq a:x$ and $\leq b:x$. Note that $a \leq b$, otherwise we could apply Rule 1. In case no unit clause $\geq a:x$ exists (resp. no unit clause $\leq b:x$ exists), a is the lowest (resp. b is the greatest) value of N. Then, we consider three cases:

1. If no binary clause exists, v can assign to x any value in the set $[a,b]$.

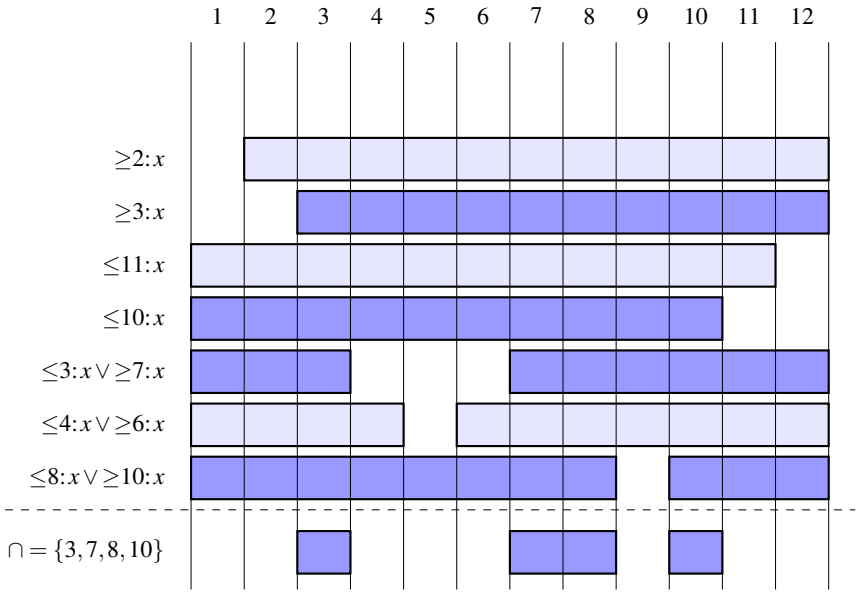


Figure 4.3: Example of a saturated multiset of regular clauses that only contain variable x, with $N = \{1,\dots,12\}$. The values of x that each clause allows are painted in blue. The subsumed clauses are painted in light blue. The bottom row shows the intersection of the values allowed by the clauses.

2. If there is exactly one binary clause of the form $\leq i : x \vee \geq j : x$, then $a \leq i$ and $b \geq j$ by definition of saturation. Therefore, v can assign to x any value in the non-empty set $[a,i] \cup [j,b]$.

3. If there is more than one binary clause, as the formula is saturated w.r.t. x, any two binary clauses in D' of the form $c_s = \leq i : x \vee \geq j : x$ and $c_t = \leq k : x \vee \geq l : x$ satisfy one of the two following restrictions: (i) $i \leq k < l \leq j$ or (ii) $i < j \leq k < l$. As we do not have tautological clauses, it always holds that $i < j$ and $k < l$. Restriction (i) implies that clause c_s subsumes c_t. From now on, we consider that all subsumed clauses are removed, and hence only restriction (ii) can hold between two binary clauses. Therefore, we can establish a total ordering $c_1 \prec \cdots \prec c_n$ on the binary clauses in D' such that, for any two clauses c_s and c_{s+1} of the form $\leq i : x \vee \geq j : x \vee A$ and $\leq k : x \vee \geq l : x \vee B$, it holds that $j \leq k$. By transitivity, this relation also happens between any two clauses c_s and c_t such that $s < t$. Note also that $c_1 = \leq i : x \vee \geq j : x$ satisfies that $a \leq i$; otherwise, we could apply Rule 2. Similarly, $c_n = \leq i : x \vee \geq j : x$ satisfies that $b \geq j$; otherwise, we could apply Rule 3. Let c_s and c_{s+1} be any two consecutive clauses of the sequence. Clearly, these two clauses allow us to assign a non-empty set of values to x. Specifically, x can take any value on the interval $[j,k] \subset [a,b]$. Note also that any clause to the left of c_s also admits the values in $[j,k]$, and the same happens with any clause to the right of c_{s+1}. Therefore, letting clause c_s be of the form

$\leq i_s : x \vee \geq j_s : x$, the values that v can assign to x are:

$$[a, i_1] \cup \bigcup_{1 \leq s \leq n-1} [j_s, i_{s+1}] \cup [j_n, b]$$

Thus, if we extend v by assigning to x any value of the indicated non-empty set, v satisfies D' and C. □

In the proof, we used an abuse of notation. Whenever we write an interval $[i, j]$, we are actually referring to the set $[i, j] \cap N$. Note that the clauses in the example of Figure 4.3 serve as an example of the set D' in the proof of Lemma 4.2.5.

Next, we proceed to prove the completeness of the Regular MaxSAT resolution calculus, as stated in Theorem 4.2.9. Before that, we introduce some necessary notation and auxiliary lemmas.

Definition 4.2.6. *The scope of a regular clause c, denoted by* $scope(c)$*, is the set of variables appearing in c. The scope of a multiset C of regular clauses is the set of variables appearing in C.*

Definition 4.2.7. *Given a regular clause c and a set X of variables such that* $X \supseteq scope(c)$*,* $score(c, X)$ *denotes the number of assignments over X that falsify c. Given a multiset C of regular clauses,* $score(C, X)$ *denotes the sum of the scores of the clauses in C.*

When clear from the context, such as when X corresponds to $scope(C)$ given a CNF formula C, we write $score(C)$.

Lemma 4.2.8. *Let C be a multiset of regular clauses. The iterative application of Rules 1–4 for resolving on variable x always terminates regardless of the order of application; i.e., it reaches a state where no rule can be applied to any two clauses of C. Hence, the result is a multiset of regular clauses that is saturated w.r.t. x.*

Proof. In order to unify the proof for the four rules, we use signed literal notation and follow the proof in [3]. In particular, considering a truth value set $N = \{1, \ldots, |N|\}$, a disjunction of regular literals $\leq i : x \vee \geq j : x$ can be denoted by $\{1, \ldots, i, j, \ldots, |N|\}$:$x$. The general idea of the proof is that, given a formula $C = F \cup \{c_\alpha, c_\beta\}$ that is not saturated w.r.t. variable x, the application of any of the Rules 1–4 over c_α, c_β results in a formula $C' = F \cup \{c_\cap, c_\cup\} \cup C_\alpha \cup C_\beta$ that is closer to a saturation w.r.t. variable x. Let us partition the clauses of C into a sequence of $|N|$ multisets $B{=}B_0, B_1, \ldots, B_{|N|-1}$, such that B_i contains all clauses from C of the form $S{:}x \vee A$ such that $|S|{=}i$. B_0 is the multiset of clauses that do not contain variable x. If $M{=}score(C)$, then $score(B_i) \leq M$ for $i \in 0, \ldots, |N|{-}1$. Let us consider $score(B_0)score(B_1)\ldots score(B_{|N|-1})$ as a word of length $|N|$ and base $M{+}1$, where $score(B_0)$ is the most significant digit. By Theorem 4.2.3, the preservation of the number of unsatisfied clauses ensures $score(C){=}score(C')$, and therefore that no digit reaches a value greater than M after applying a rule. However, we will show that the value of the $|N|$-digit word always increases under any rule, thereby completing the proof, since this word has a finite maximum value. When we apply one of Rules 1–4, we remove clauses $c_\alpha{=}S_\alpha{:}x \vee A$ and $c_\beta{=}S_\beta{:}x \vee B$ from their corresponding multisets $B_{|S_\alpha|}$, $B_{|S_\beta|}$. We also add a number of new clauses to some multisets of

B, including clause $c_\cap{=}S_\cap{:}x \vee A \vee B$. By the definition of the rules, $S_\cap \subset S_\alpha, S_\beta$ (see also Figure 4.2), hence $|S_\cap|<|S_\alpha|, |S_\beta|$ and also $score(c_\alpha)>0$. Therefore, $c_\cap$ will be inserted into a digit of B more significant than the digits of the removed clauses c_α, c_β, thus increasing the value of the word. □

Theorem 4.2.9. *For any multiset C of regular clauses, it holds that*

$$C \vdash \underbrace{\square, \ldots, \square}_{m}, D$$

where D is a satisfiable multiset of regular clauses, m is the minimum number of clauses of C falsified by any assignment, and $C \vdash \square, \ldots, \square, D$ denotes that the multiset of clauses $\{\square, \ldots, \square, D\}$ can be obtained from the multiset C by applying the Regular MaxSAT resolution rule a finite number of times.

Proof. Let $x_1, \ldots, x_n$ be any list of the variables occurring in C. We construct two sequences of multisets $C_0, \ldots, C_n$ and $D_1, \ldots, D_n$ so that $C = C_0$; for $i = 1, \ldots, n$, $C_i \cup D_i$ is a saturation of C_{i-1} w.r.t. x_i; and, for $i = 1, \ldots, n$, C_i is the multiset of clauses not containing variables $x_1, \ldots, x_i$ and D_i is the multiset of clauses containing variable x_i. These sequences can be computed in a finite number of steps because C_{i+1} contains one variable less than C_i, and by Lemma 4.2.8 the saturation of C_i can be computed in a finite number of steps. For $i = 1, \ldots, n$, we saturate C_{i-1} w.r.t. x_i, and then partition the resulting multiset into a subset D_i containing x_i, and another subset C_i not containing x_i. Since C_n contains no variables, it is either an empty multiset or just contains a finite number of empty clauses $\{\square, \ldots, \square\}$.

We will define a complete satisfying assignment for the multiset $D = \bigcup_{i=1}^{n} D_i$ to prove that it is satisfiable. For $i = 1, \ldots, n$, let $d_i = D_i \cup \cdots \cup D_n$ and let $d_{n+1} = \emptyset$. Notice that d_i only contains the variables in $\{x_i, \ldots, x_n\}$, $d_i = D_i \cup d_{i+1}$ and d_i is saturated w.r.t. x_i because d_{i+1} does not contain x_i and $C_i \cup D_i$ is saturated w.r.t. x_i.

We now define a sequence of assignments $v_1, \ldots, v_{n+1}$, where v_{n+1} is the empty assignment and, therefore, satisfies d_{n+1}. Assignment v_i is defined from assignment v_{i+1} as follows: Assume by the induction hypothesis that v_{i+1} satisfies d_{i+1}. Since d_i is saturated w.r.t. x_i and $d_i = D_i \cup d_{i+1}$, by Lemma 4.2.5, we can extend v_{i+1} with a value of x_i that satisfies d_i. Iterating, we have that v_1 satisfies $d_1 = D = \bigcup_{i=1}^{n} D_i$.

Since the soundness theorem ensures that the regular MaxSAT resolution rules preserve the number of unsatisfied clauses for every assignment, the number of empty clauses in C_n ($|C_n| = m$) is the minimum number of clauses of C that are unsatisfied by any assignment. In particular, assignment v_1 falsifies exactly m clauses of C. □

4.3 A Variable Elimination Algorithm for Regular MaxSAT

Algorithm 4.1 shows the pseudo-code of a variable elimination algorithm derived from the proof of Theorem 4.2.9: Given an input multiset of regular clauses C with n different variables, the algorithm returns the minimum number m of clauses of C that will

Algorithm 4.1: An exact variable elimination algorithm for Regular MaxSAT

Input: C: a multiset of regular clauses.
Output: (m, I): an assignment v that falsifies the minimum number of clauses (m).

$C_0 \leftarrow C$;
for $i \leftarrow 1$ **to** n **do**
  $C' \leftarrow saturation(C_{i-1}, x_i)$;
  $\langle C_i, D_i \rangle \leftarrow partition(C', x_i)$;
$m \leftarrow |C_n|$;
$I \leftarrow \emptyset$;
for $i \leftarrow n$ **downto** 1 **do**
  $I \leftarrow I \cup [x_i \mapsto max_extension(x_i, I, D_i)]$;
return (m, I)

be unsatisfied for any assignment, and an optimal assignment I that falsifies exactly m clauses of C.

The variable elimination algorithm has the following functions:

- Function $saturation(C, x)$ computes a saturation of C w.r.t. x. By Lemma 4.2.8, this can be implemented by systematically trying to apply Rules 1–4 to any pair of clauses containing x, until no application can be done.
- Function $partition(C, x_i)$ partitions C into two multisets, C_i and D_i so that C_i contains the regular clauses without occurrences of variable x_i, and D_i contains the regular clauses with occurrences of x_i.
- Function $max_extension(x_i, I, D_i)$ computes a truth assignment I for x_i as follows: if v satisfies all the clauses in D_i, including the case in which $D_i = \{\}$, then the function returns the greatest truth value. Otherwise, by Lemma 4.2.5, I assigns to x_i one value of the non-empty set $[a, i_1] \cup \bigcup_{1 \leq s \leq n-1} [j_s, i_{s+1}] \cup [j_n, b]$.

The algorithm has two parts. In the first part, the algorithm successively saturates w.r.t. all the variables occurring in the input multiset. Once the current multiset is saturated w.r.t. a variable x_i, it partitions the resulting multiset into two multisets: C_i and D_i. The multiset C_i contains the clauses without occurrences of x_i, and the multiset D_i contains the clauses with occurrences of x_i. The algorithm continues saturating C_i w.r.t. one of the remaining variables, and ignores D_i. This process continues until all the variables are eliminated. At the end, C_n does not contain any variable, and the number of empty clauses in C_n is the returned minimum number of unsatisfied clauses. In the second part, the algorithm builds an optimal assignment as function $max_extension(x_i, I, D_i)$ states.

Example 4.3.1. *Let* $N = \{1,2,3,4,5\}$ *and let* $C_0 = \{\leq 1 : x_3, \geq 3 : x_1 \vee \geq 2 : x_2, \leq 1 : x_1 \vee \geq 2 : x_2, \leq 2 : x_2 \vee \geq 2 : x_3, \geq 1 : x_2 \vee \leq 4 : x_3, \geq 3 : x_3\}$. *Saturating* C_0 *w.r.t.* x_1, *we get* $C_1 = \{\leq 1 : x_3, \geq 2 : x_2, \leq 2 : x_2 \vee \geq 2 : x_3, \geq 1 : x_2 \vee \geq 4 : x_3, \geq 3 : x_3\}$ *and* $D_1 = \{\leq 1 : x_1 \vee$

Rule 1

$$\begin{array}{l} c_\alpha : (\geq j : x \vee A, u) \\ c_\beta : (\leq k : x \vee B, w) \\ \hline c_\cap : (A \vee B, m) \\ c_\cup : (\leq k : x \vee \geq j : x \vee A \vee B, m) \\ C_\alpha : (\geq j : x \vee A \vee \neg B, m) \\ C_\beta : (\leq k : x \vee \neg A \vee B, m) \\ c'_\alpha : (\geq j : x \vee A, u - m) \\ c'_\beta : (\leq k : x \vee B, w - m) \end{array}$$

provided that $k < j$,
$A \vee B$ is not a tautology
and $m = \min(u, w)$

Rule 2

$$\begin{array}{l} c_\alpha : (\geq j : x \vee A, u) \\ c_\beta : (\leq k : x \vee \geq l : x \vee B, w) \\ \hline c_\cap : (\geq l : x \vee A \vee B, :) \\ c_\cup : (\leq k : x \vee \geq j : x \vee A \vee B, m) \\ C_\alpha : (\geq j : x \vee A \vee \neg B, m) \\ C_\beta : (\leq k : x \vee \geq l : x \vee \neg A \vee B, m) \\ c'_\alpha : (\geq j : x \vee A, u - m) \\ c'_\beta : (\leq k : x \vee \geq l : x \vee B, w - m) \end{array}$$

provided that $k < j < l$
$A \vee B$ is not a tautology
and $m = \min(u, w)$

Rule 3

$$\begin{array}{l} c_\alpha : (\leq i : x \vee \geq j : x \vee A, u) \\ c_\beta : (\leq k : x \vee B, w) \\ \hline c_\cap : (\leq i : x \vee A \vee B, :) \\ c_\cup : (\leq k : x \vee \geq j : x \vee A \vee B, m) \\ C_\alpha : (\leq i : x \vee \geq j : x \vee A \vee \neg B, m) \\ C_\beta : (\leq k : x \vee \neg A \vee B, m) \\ c'_\alpha : (\leq i : x \vee \geq j : x \vee A, u - m) \\ c'_\beta : (\leq k : x \vee B, w - m) \end{array}$$

provided that $i < k < j$
$A \vee B$ is not a tautology
and $m = \min(u, w)$

Rule 4

$$\begin{array}{l} c_\alpha : (\leq i : x \vee \geq j : x \vee A, u) \\ c_\beta : (\leq k : x \vee \geq l : x \vee B, w) \\ \hline c_\cap : (\leq i : x \vee \geq l : x \vee A \vee B, m) \\ c_\cup : (\leq k : x \vee \geq j : x \vee A \vee B, m) \\ C_\alpha : (\leq i : x \vee \geq j : x \vee A \vee \neg B, m) \\ C_\beta : (\leq k : x \vee \geq l : x \vee \neg A \vee B, m) \\ c'_\alpha : (\leq i : x \vee \geq j : x \vee A, u - m) \\ c'_\beta : (\leq k : x \vee \geq l : x \vee B, w - m) \end{array}$$

provided that $i < k < j < l$
$A \vee B$ is not a tautology
and $m = \min(u, w)$

Figure 4.4: Proof system for Weighted Regular MaxSAT.

$\geq 3 : x_1 \vee \geq 2 : x_2\}$. *Saturating* C_1 *w.r.t.* x_2*, we get* $C_2 = \{\leq 1 : x_3, \geq 3 : x_3\}$ *and* $D_2 = \{\geq 2 : x_2, \leq 2 : x_2 \vee \geq 2 : x_3, \geq 1 : x_2 \vee \geq 4 : x_3\}$. *Saturating* C_2 *w.r.t.* x_3*, we get* $C_3 = \{\square\}$ *and* $D_3 = \{\leq 1 : x_3 \vee \geq 3 : x_3\}$. *Hence, the minimum number of unsatisfied clauses is 1, and* $x_3 \mapsto 5, x_2 \mapsto 5, x_1 \mapsto 5$ *is an optimal assignment.*

4.4 Weighted Regular MaxSAT Resolution

For ease of presentation, we have presented the inference rules for (unweighted) Regular MaxSAT. The weighted version of the inference rules of the proposed Regular MaxSAT proof systems is shown in Figure 4.4. Conclusions with weight zero are omitted.

From a conceptual viewpoint, a clause with weight w is equivalent to having w copies of that clause. Thus, the application of a weighted rule collapses $\min(u, w)$

applications of the unweighted rule. Since the premises can have different weights, we add again the premises with weights $u - \min(u,w)$ and $w - \min(u,w)$. One of such weights is zero, and the other allows the use of the copies of the premise that have not been consumed in other inference steps. So, the results of soundness and completeness for the weighted case follow directly from the results in the unweighted case. Nevertheless, in practice, it is much more efficient to apply the weighted rules when we have weighted clauses.

4.5 Conclusions

We have defined the first resolution proof system for Regular MaxSAT and proved its soundness and completeness; described the first exact variable elimination for Regular MaxSAT that incorporates a notion of saturation exploiting the structure of regular signs; and extended the proposed proof system to solve Weighted Regular MaxSAT.

Chapter 5

Tableaux for Regular MaxSAT and Regular MinSAT

This chapter introduces novel tableau calculi for solving the MaxSAT and MinSAT problems within the framework of regular propositional logic, establishing their soundness and completeness for both Weighted Partial Regular MaxSAT and Weighted Partial Regular MinSAT. Additionally, it proposes novel Mixed-Integer Linear Programming (MILP) encodings that enable state-of-the-art MILP solvers to efficiently handle these weighted partial optimization problems. Finally, it reports an empirical evaluation of the proposed MILP-based approach on randomly generated Max-2SAT and Max-3SAT instances.

5.1 Introduction

Regular logic generalizes Boolean logic by considering a totally ordered set of truth values N, which can be finite or infinite, and associating each formula with a regular sign. These signs are restricted to subsets of the form $\{j \in N \mid j \geq i\}$, denoted by $\geq i$, or $\{j \in N \mid j \leq i\}$, denoted by $\leq i$, for some $i \in N$. Within this framework, an assignment is a mapping that assigns a truth value from N to each propositional variable, which is extended to arbitrary formulas by interpreting conjunction, disjunction, and negation through the minimum, maximum, and Łukasiewicz negation functions, respectively. An assignment v satisfies a regular formula $S : A$ if and only if $v(A) \in S$. Given a multiset of such formulas, the Regular MaxSAT problem aims to find an assignment that maximizes the number of satisfied formulas.

The tableau method is a systematic procedure for solving satisfiability problems, often viewed as a proof by case distinction. The tableau calculus for SAT provides a systematic procedure for determining the satisfiability of propositional formulas through semantic tree construction and branch closure analysis [37, 89]. However, a critical challenge arises when attempting to apply classical tableau methods to MaxSAT and MinSAT: proof systems designed for SAT are unsound for these optimization prob-

lems. The fundamental issue is that classical tableau calculi preserve satisfiability between premises and conclusions, but do not preserve the number of satisfied/unsatisfied formulas, which is essential for MaxSAT and MinSAT [64].

This shortcoming has motivated the development of specialized tableau calculi that are sound and complete for Boolean MaxSAT and MinSAT (e.g. [7, 44, 45, 69, 71]). Building on this line of research, this chapter introduces new tableau calculi for solving the MaxSAT and MinSAT problems within regular propositional logic, and establishes their soundness and completeness for the Weighted Partial MaxSAT and Weighted Partial MinSAT variants. Since a direct implementation of these calculi is not computationally efficient, this chapter also investigates Mixed-Integer Linear Programming (MILP) encodings that follow their underlying principles and allow state-of-the-art MILP solvers to handle these weighted partial optimization problems more effectively. Finally, it reports an empirical evaluation of the proposed MILP-based approach on randomly generated Max-2SAT and Max-3SAT instances.

After the publication of our results on Weighted Partial Regular MaxSAT and MinSAT tableaux in [31, 33, 34, 35], Fiorino later proposed related tableau calculi for these problems in [46].

The remainder of this chapter is structured as follows. Section 5.2 presents a tableau calculus for Regular MaxSAT. Section 5.3 introduces its extension to the weighted partial setting and establishes its soundness and completeness. Section 5.4 develops a tableau calculus for Weighted Partial Regular MinSAT and proves its soundness and completeness. Section 5.5 presents tableau calculi for Regular MaxSAT and Regular MinSAT for instances in clausal form. Section 5.6 introduces a reduction from Regular MaxSAT and Regular MinSAT to MILP and evaluates it experimentally on randomly generated Max-2SAT and Max-3SAT instances. Finally, Section 5.7 concludes the chapter.

5.2 A Regular MaxSAT tableau calculus

This section presents a sound and complete tableau proof system for Regular MaxSAT. We present here a calculus for unweighted Regular MaxSAT and extend it to Weighted Partial MaxSAT in Section 5.3 to make the introduction to many-valued tableaux easier. The rest of the calculi of this chapter are directly presented for Weighted Partial MaxSAT/MinSAT.

Definition 5.2.1. *A tableau is a rooted tree with a finite number of branches whose nodes are labelled by either a regular formula or a box ($\Box$). A box in a tableau denotes a contradiction. A branch is a maximal path in a tree, and we assume that branches have a finite number of nodes.*

Definition 5.2.2. *Let $N = \{0, \frac{1}{n-1}, \ldots, \frac{n-1}{n-1}\}$ be a truth value set and let $\Phi = \{A_1, \ldots, A_m\}$ be a multiset of regular formulas. A tableau is constructed by a sequence of applications of the following rules:*

- **Initialize** *We start by generating a tree with a single branch with m nodes such that each node is labelled with a formula of Φ. Such a tableau is a Regular*

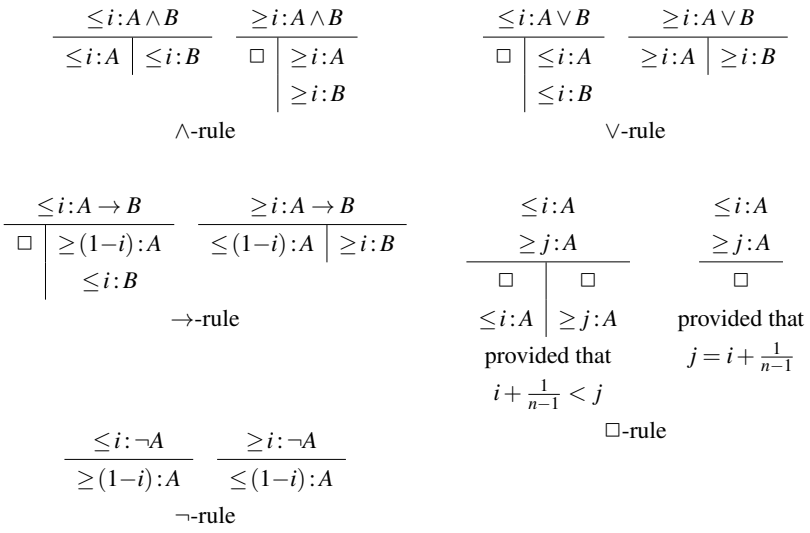


Figure 5.1: Tableau expansion rules for Regular MaxSAT.

MaxSAT tableau for Φ, called the initial tableau. All its formulas are declared to be active.

- **$\wedge$-rule:** *Given a tableau T for Φ and a branch b of T, if b contains an active regular formula $\leq i:A \wedge B$ (resp. $\geq i:A \wedge B$), the tableau obtained by appending a new left branch with a node below b labelled with $\leq i:A$ (resp. $\Box$) and a new right branch with a node (resp. two nodes) below b labelled with $\leq i:B$ (resp. $\geq i:A$ and $\geq i:B$) is a tableau for Φ. Regular formula $\leq i:A \wedge B$ (resp. $\geq i:A \wedge B$) becomes inactive in b and the added regular formulas are declared to be active.*

- **$\vee$-rule:** *Given a tableau T for Φ and a branch b of T, if b contains an active regular formula $\leq i:A \vee B$ (resp. $\geq i:A \vee B$), the tableau obtained by appending a new left branch with a node below b labelled with $\Box$ (resp. $\geq i:A$) and a new right branch with two nodes (resp. a node) below b labelled with $\leq i:A$ and $\leq i:B$ (resp. $\geq i:B$) is a tableau for Φ. Regular formula $\leq i:A \vee B$ (resp. $\geq i:A \vee B$) becomes inactive in b and the added regular formulas are declared to be active.*

- **$\to$-rule:** *Given a tableau T for Φ and a branch b of T, if b contains an active regular formula $\leq i:A \to B$ (resp. $\geq i:A \to B$), the tableau obtained by appending a new left branch with a node below b labelled with $\Box$ (resp. $\leq (1-i):A$) and a new right branch with two nodes (resp. a node) below b labelled with $\geq (1-i):A$ and $\leq i:B$ (resp. $\geq i:B$) is a tableau for Φ. Regular formula $\leq i:A \to B$ (resp. $\geq i:A \to B$) becomes inactive in b and the added regular formulas are declared to be active.*

- **$\neg$-rule:** *Given a tableau T for Φ and a branch b of T, if b contains an active regular formula $\leq i:\neg A$ (resp. $\geq i:\neg A$), the tableau obtained by appending a*

node below b labelled with $\geq(1-i):A$ (resp. $\leq(1-i):A$) is a tableau for Φ. Regular formula $\leq i:\neg A$ (resp. $\geq i:\neg A$) becomes inactive in b and the added regular formula is declared to be active.

- **$\Box$-rule:** *Given a tableau T for Φ and a branch b of T, if b contains two active regular formulas $\leq i:A$ and $\geq j:A$ with $i+\frac{1}{n-1}<j$ (resp. $j=i+\frac{1}{n-1}$), the tableau obtained by appending a new left branch with two nodes below b labelled with $\Box$ and $\leq i:A$ and a new right branch with two nodes below b labelled with $\Box$ and $\geq j:A$ (resp. appending a node below b labelled with $\Box$) is a tableau for Φ. Regular formulas $\leq i:A$ and $\geq j:A$ become inactive in b and the added regular formulas are declared to be active.*

Definition 5.2.2 formally introduces the inference rules of the calculus, which are also referred to as the expansion rules. Their graphical representation is provided in Figure 5.1. Throughout the remainder of the thesis, we often omit their formal definitions and present only their graphical representations when introducing a tableau calculus. Although the rules are defined assuming a finite truth value set, the calculus can be readily adapted to operate over the unit interval.

Definition 5.2.3. *Let T be a tableau for a multiset Φ of regular formulas. A branch b of T is saturated when no further expansion rules can be applied to b, and T is completed when all its branches are saturated. The cost of a saturated branch is the number of boxes in the branch. The cost of a completed tableau is the minimum cost among all its branches.*

Theorem 5.2.4. Soundness & completeness. *Let T be a completed Regular MaxSAT tableau for a multiset Φ of regular formulas. Then, the cost of T is m if and only if the minimum number of formulas in Φ that are falsified by any assignment is m.*

We omit the proof of soundness and completeness of Theorem 5.2.4 because these properties follow from the soundness and completeness theorem for Weighted Partial MaxSAT in Section 5.3.1.

Example 5.2.5. *Figure 5.2 illustrates the construction of a tableau, assuming $N=\{0,\frac{1}{3},\frac{2}{3},1\}$ and utilizing the preceding calculus, to determine the minimum number of unsatisfied regular formulas in the multiset of regular formulas $\Phi=\{\geq 1:x_1\wedge x_3,\leq\frac{1}{3}:x_1\wedge x_2,\geq\frac{1}{3}:\neg x_3\}$. The initial tableau, presented in the leftmost position at the top, comprises one node for each regular formula in Φ. The second tableau shows the application of the $\neg$-rule to $\geq\frac{1}{3}:\neg x_3$, and after that, the application of the $\wedge$-rule to $\leq\frac{1}{3}:x_1\wedge x_2$. The third tableau is the result of applying the following expansion rules: the $\wedge$-rule to $\geq 1:x_1\wedge x_3$ on both branches, which results in a total of four branches; the $\Box$-rule to $\geq 1:x_1$ and $\leq\frac{1}{3}:x_1$ on the second leftmost branch, and the $\Box$-rule to $\leq\frac{2}{3}:x_3$ and $\geq 1:x_3$ on the second, third and fifth branches (from left to right) as presented in the final tableau. Since the tableau is completed and the minimum number of boxes among its branches is one, the minimum number of unsatisfied regular formulas in Φ is also one.*

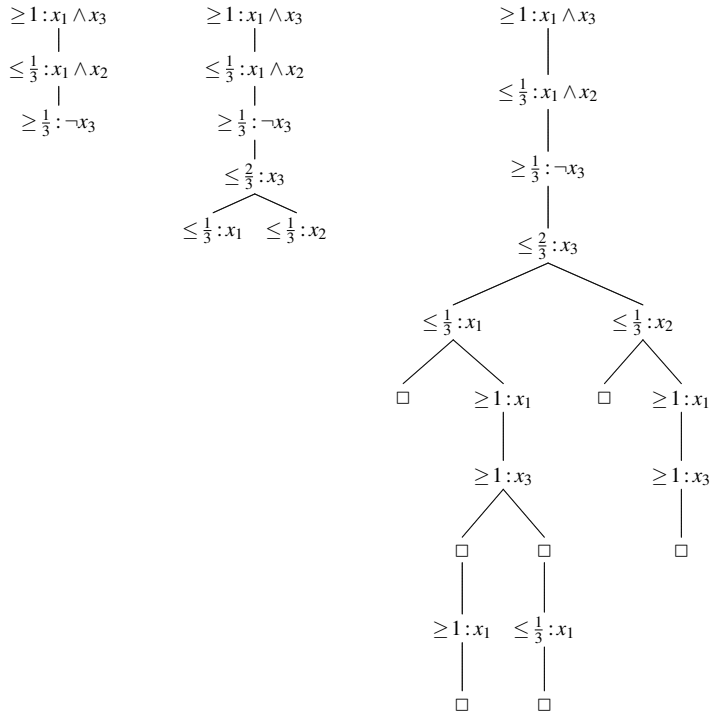


Figure 5.2: A completed tableau for the Regular MaxSAT instance $\Phi = \{\geq 1 : x_1 \wedge x_3, \leq \frac{1}{3} : x_1 \wedge x_2, \geq \frac{1}{3} : \neg x_3\}$.

Remark 5.2.6. *Throughout this thesis we usually prove the soundness for the functionally complete set $\{\neg, \vee\}$, since all other connectives are definable in terms of these two: $A \rightarrow B$ is equivalent to $\neg A \vee B$, and $A \wedge B$ is equivalent to $\neg(\neg A \vee \neg B)$. Soundness for the $\rightarrow$-rule and $\wedge$-rule then follows immediately from the soundness of the $\neg$-rule and $\vee$-rule by substitution.*

Example 5.2.7. *To demonstrate the derivation of the $\wedge$-rule from the $\neg$- and $\vee$-rules, we consider the formula $\phi = \leq i : \neg(\neg A \vee \neg B)$ as shown in Figure 5.3.*

5.3 A Weighted Partial Regular MaxSAT Tableau Calculus

This section presents a sound and complete tableau calculus for Weighted Partial Regular MaxSAT and establishes its soundness and completeness. In the input multiset of formulas Φ, each formula is represented as a pair (ϕ, w), where ϕ is a regular formula and w is its associated weight. We distinguish between two classes of formulas based

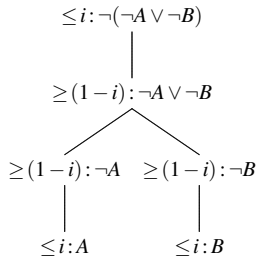


Figure 5.3: Derivation of the $\wedge$-rule from the $\neg$- and $\vee$-rules.

on their weight: if $w = \infty$, the formula is classified as a hard formula, representing a constraint that cannot be violated. Conversely, if w is a positive integer, it is classified as a soft formula, representing an optimization preference. An assignment is feasible if it satisfies all hard formulas.

The expansion rules of the proposed calculus for Weighted Partial Regular MaxSAT are shown in Figure 5.4, where the symbol $\square$ denotes a contradiction and the symbol $\blacksquare$ denotes an infeasible solution. They are used to construct a tableau from an initial single branch consisting of as many nodes as there are weighted regular formulas in the input multiset, each of which is declared to be active. When a rule is applied, the soft premises become inactive and the conclusions become active in the branch where the rule is applied. The corresponding notion of cost is defined as follows:

Definition 5.3.1. *Let T be a tableau for a Weighted Partial Regular MaxSAT instance Φ. A branch b of T is saturated when no further expansion rules can be applied to b, and T is completed when all its branches are saturated. The cost of a saturated branch is infinite if it contains the symbol $\blacksquare$; otherwise, it is the sum of the weights of the soft regular formulas containing the symbol $\square$. The cost of a tableau is the minimum cost among all its branches. If the cost of a tableau is infinite, there is no feasible solution.*

Dealing with weighted soft regular premises is similar to the unweighted case. When there is exactly one premise, the expansion rules are similar to the unweighted ones except for the fact that they propagate the weight from the premise to the conclusions. Note that whether a given assignment satisfies a soft regular formula does not depend on its weight. The weight is the penalty to be paid if the formula is not satisfied.

The only rule with two soft premises is the $\square$-rule. In this case, we distinguish two cases when both premises are soft regular formulas:

1. the expansion rule with proviso $j = i + \frac{1}{n-1}$: exactly one of the premises is not satisfied, because the union of the regular signs is the entire truth value set and their intersection is empty.

2. the expansion rule with proviso $j > i + \frac{1}{n-1}$: one premise is necessarily not satisfied, but both premises are not satisfied when the value assigned to A is smaller than j and greater than i.

$$\frac{(\leq i : A \wedge B, w)}{(\leq i : A, w) \mid (\leq i : B, w)} \qquad \frac{(\geq i : A \wedge B, w)}{(\Box, w) \mid \begin{array}{l}(\geq i : A, w)\\(\geq i : B, w)\end{array}} \qquad \frac{(\leq i : A \wedge B, \infty)}{(\leq i : A, \infty) \mid (\leq i : B, \infty)} \qquad \frac{(\geq i : A \wedge B, \infty)}{\begin{array}{l}(\geq i : A, \infty)\\(\geq i : B, \infty)\end{array}}$$

$\wedge$-rule

$$\frac{(\leq i : A \vee B, w)}{(\Box, w) \mid \begin{array}{l}(\leq i : A, w)\\(\leq i : B, w)\end{array}} \qquad \frac{(\geq i : A \vee B, w)}{(\geq i : A, w) \mid (\geq i : B, w)} \qquad \frac{(\leq i : A \vee B, \infty)}{\begin{array}{l}(\leq i : A, \infty)\\(\leq i : B, \infty)\end{array}} \qquad \frac{(\geq i : A \vee B, \infty)}{(\geq i : A, \infty) \mid (\geq i : B, \infty)}$$

$\vee$-rule

$$\frac{\begin{array}{l}(\leq i : A, w_1)\\(\geq j : A, w_2)\end{array}}{(\Box, w_1) \mid \begin{array}{l}(\Box, w_2)\\(\leq i : A, w_1)\end{array}} \text{ provided that } i + \tfrac{1}{n-1} < j \qquad \frac{\begin{array}{l}(\leq i : A, w_1)\\(\geq j : A, w_2)\end{array}}{\begin{array}{l}(\Box, w)\\(\leq i : A, w_1 - w)\\(\geq j : A, w_2 - w)\end{array}} \text{ provided that } j = i + \tfrac{1}{n-1} \text{ where } w = \min(w_1, w_2)$$

$$\frac{\begin{array}{l}(\leq i : A, w)\\(\geq j : A, \infty)\end{array}}{\begin{array}{l}(\Box, w)\\(\geq j : A, \infty)\end{array}} \text{ provided that } i < j \qquad \frac{\begin{array}{l}(\leq i : A, \infty)\\(\geq j : A, w)\end{array}}{\begin{array}{l}(\Box, w)\\(\leq i : A, \infty)\end{array}} \text{ provided that } i < j$$

$\Box$-rule

$$\frac{\begin{array}{l}(\leq i : A, \infty)\\(\geq j : A, \infty)\end{array}}{\blacksquare} \text{ provided that } i < j$$

$\blacksquare$-rule

$$\frac{(\leq i : \neg A, w)}{(\geq (1-i) : A, w)} \qquad \frac{(\geq i : \neg A, w)}{(\leq (1-i) : A, w)} \qquad \frac{(\leq i : \neg A, \infty)}{(\geq (1-i) : A, \infty)} \qquad \frac{(\geq i : \neg A, \infty)}{(\leq (1-i) : A, \infty)}$$

$\neg$-rule

Figure 5.4: Tableau expansion rules for Weighted Partial Regular MaxSAT.

When there is a hard premise and a soft premise in the $\Box$-rule, since the intersection of the signs of the premises is empty, we derive a contradiction whose weight matches that of the soft formula, while the hard formula remains active. For this reason, the hard formula is repeated in the conclusion.

The $\blacksquare$-rule prunes the branch due to the detection of an infeasible solution. It detects the violation of a hard constraint.

5.3.1 Soundness & completeness

We prove the soundness and completeness of the Weighted Partial Regular MaxSAT calculus presented in Figure 5.4. Prior to establishing the completeness theorem, we prove termination and the soundness of the expansion rules.

Proposition 5.3.2. *A Weighted Partial Regular MaxSAT tableau for a multiset Φ of weighted regular formulas is completed in a finite number of steps.*

Proof. We first create an initial tableau and then apply expansion rules in the newly created branches until they become saturated. The $\vee$- and $\neg$-rule reduce the number of connectives. Since the number of connectives in the initial formula is finite and no rule introduces new connectives, these rules can only be applied a finite number of times. The $\Box$-rule produces a formula with the symbol $\Box$, which cannot be a premise of any rule, and another formula that can be a premise of another rule. Since the number of potential premises is reduced, it can only be applied a finite number of times. Observe that, in the rule with proviso $j = i + \frac{1}{n-1} < j$, at least one of the conclusions has weight zero and is not added. The $\blacksquare$-rule has two hard formulas as premises and produces a single symbol $\blacksquare$ in the conclusion, which cannot be a premise of any rule, and therefore it can only be applied a finite number of times. In addition, The $\blacksquare$-rule prunes the branch. Hence, the construction of any completed tableau for Weighted Partial Regular MaxSAT terminates in a finite number of steps. $\Box$

Proposition 5.3.3. *The $\vee$-, $\neg$-, $\blacksquare$- and $\Box$-rules of the Weighted Partial Regular MaxSAT calculus in Figure 5.4 are sound in the following sense: for any assignment v, the sum of the weights of the unsatisfied soft premises is preserved in at least one branch and does not decrease in the remaining branches. Furthermore, rules with hard premises preserve the satisfiability from the premises to the conclusions in at least one branch.*

Proof. To prove the soundness of the $\vee$-rule with a single soft premise of weight w, we must demonstrate that if an assignment v falsifies the premise, there is one branch in the conclusions where v falsifies one formula, and v falsifies at least one formula in the rest of the branches. Besides, if v satisfies the premise, v satisfies all formulas in at least one branch. We distinguish the following cases: (i) the premise is $(\leq i : A \vee B, w)$: the rule produces two branches: a left branch containing $(\Box, w)$, and a right branch containing $(\leq i : A, w)$ and $(\leq i : B, w)$. If v satisfies the premise, then $v(A \vee B) = \max(v(A), v(B)) \leq i$, so $v(A) \leq i$ and $v(B) \leq i$. Hence, v satisfies both formulas in the right branch with cost 0, while the left branch has cost w. If v does not satisfy the premise (cost w), then $\max(v(A), v(B)) > i$. In the left branch, $(\Box, w)$ has cost w, preserving the cost. In the

right branch, at least one of $\leq i{:}A$ or $\leq i{:}B$ is unsatisfied, so the cost is at least w; (ii) the premise is $(\geq i{:}A \vee B, w)$: the rule produces two branches: $\geq i{:}A, w$ and $\geq i{:}B, w$. If v satisfies the premise, then $\max(v(A), v(B)) \geq i$, so $v(A) \geq i$ or $v(B) \geq i$, and at least one branch has cost 0. If v does not satisfy the premise, then $v(A) < i$ and $v(B) < i$, so both conclusions are unsatisfied with cost w, preserving the total weight of unsatisfied formulas; and (iii) the premise is a hard formula: the rules preserve satisfiability and are identical to those defined for Regular SAT tableaux [49].

To prove the soundness of the $\neg$-rule, we must consider that, when the premise is $(\leq i{:}\neg A, w)$, an assignment v satisfies $\leq i{:}\neg A$ if and only if $v(\neg A) = 1 - v(A) \leq i$ if and only if $v(A) \geq 1 - i$, which holds if and only if v satisfies $\geq (1-i){:}A$. When the premise is $(\geq i{:}\neg A, w)$, v satisfies $\geq i{:}\neg A$ if and only if $v(\neg A) = 1 - v(A) \geq i$ if and only if $v(A) \leq 1 - i$, which holds if and only if v satisfies $\leq (1-i){:}A$. In both cases, the total weight of unsatisfied formulas is preserved because the conclusion consists of a single formula with weight w. The same reasoning applies when the premises are hard.

In the $\blacksquare$-rule, since the intersection between the signs $\leq i$ and $\geq j$ is empty, one of the premises is necessarily falsified. Consequently, the branch must be pruned, as it corresponds to an infeasible solution.

To prove the soundness of the $\square$-rule we distinguish the following cases: (i) soft premises with proviso $i + \frac{1}{n-1} < j$: the premises are $(\leq i{:}A, w_1)$ and $(\geq j{:}A, w_2)$. Since the intersection of the signs is empty, at least one premise is necessarily unsatisfied. The rule produces two branches: the left branch contains $(\square, w_1)$ and $(\geq j{:}A, w_2)$, and the right branch contains $(\square, w_2)$ and $(\leq i{:}A, w_1)$. If $v(A) \leq i$, the premise cost is w_2. In the right branch, $(\square, w_2)$ is falsified and $\leq i{:}A$ is satisfied, giving cost w_2. In the left branch, both are falsified, giving a cost of $w_1 + w_2 \geq w_2$. If $i < v(A) < j$, both premises are falsified with cost $w_1 + w_2$, and both branches also have cost $w_1 + w_2$. If $v(A) \geq j$, the premise cost is w_1. In the left branch, $(\square, w_1)$ is falsified and $\geq j{:}A$ is satisfied, giving cost w_1. In the right branch, both are falsified, giving a cost of $w_1 + w_2 \geq w_1$. (ii) soft premises with proviso $j = i + \frac{1}{n-1}$: the premises are $(\leq i{:}A, w_1)$ and $(\geq j{:}A, w_2)$, with $w = \min(w_1, w_2)$. Since $j = i + \frac{1}{n-1}$, there is no value in N strictly between i and j, so either $v(A) \leq i$ or $v(A) \geq j$. If $v(A) \leq i$, the premise cost is w_2. In the conclusion, $(\square, w)$ is falsified (cost w), $\leq i{:}A$ is satisfied, and $\geq j{:}A$ is falsified (cost $w_2 - w$), giving total cost $w + (w_2 - w) = w_2$. If $v(A) \geq j$, the premise cost is w_1. In the conclusion, $(\square, w)$ is falsified (cost w), $\leq i{:}A$ is falsified (cost $w_1 - w$), and $\geq j{:}A$ is satisfied, giving total cost $w + (w_1 - w) = w_1$. (iii) there is one soft premise with weight w and one hard premise: the rule produces a single branch containing $(\square, w)$ and the hard premise. As v must satisfy the hard premise, then the soft premise is unsatisfied because the intersection of their signs is empty (cost w), and the total weight of unsatisfied formulas is preserved. $\square$

Theorem 5.3.4. Soundness and completeness. *Let T be a completed Weighted Partial Regular MaxSAT tableau for a multiset Φ of weighted regular formulas. Then, the cost of T is m if and only if the minimum total weight of the soft formulas in Φ that are falsified by any assignment satisfying all hard formulas is m; and the cost of T is infinite if and only if no assignment satisfies all hard formulas.*

Proof. (*Soundness:*) T was derived by creating a sequence of tableaux $T_0, \ldots, T_n$ ($n \geq$

0) such that T_0 is an initial tableau for Φ, $T_n = T$, and T_i was obtained by a single application of one of the $\vee$-, $\neg$-, $\blacksquare$- and $\square$-rules on a branch of T_{i-1}, for $i = 1, \ldots, n$. By Proposition 5.3.2, we know that such a sequence is finite. Since T has cost m, T_n contains a branch b with cost m, and the rest of the branches have cost at least m. Moreover, the active formulas in every branch of T_n are of the form $(\square, w)$ or are weighted regular literals of the form $(\leq i : x, w)$ or $(\geq i : x, w)$ such that, for each propositional variable x, the intersection of the signs of all regular literals is non-empty; otherwise, we could yet apply expansion rules and T_n would not be completed.

Given a saturated branch of cost m, the assignment that sets each variable x to one value of the intersection of the signs of all active weighted regular literals containing x satisfies all hard formulas and only falsifies the formulas containing the symbol $\square$. Moreover, there cannot be any assignment satisfying all hard formulas in a branch of T_n whose total weight of unsatisfied formulas is less than m, because the total weight of unsatisfied formulas of the form $(\square, w)$ in a saturated branch is at least m. Therefore, the minimum total weight of unsatisfied formulas among all branches of T_n is m.

The minimum total weight of active unsatisfied soft formulas is preserved in the sequence of tableaux $T_0, \ldots, T_n$. Since T_0 is formed by a single branch containing only the formulas in Φ and all these formulas are active, the minimum total weight of the soft formulas that are falsified in Φ by any assignment satisfying all hard formulas is m.

If the cost of T is infinite, every branch of T_n contains the symbol $\blacksquare$. By Proposition 5.3.3, this means that no assignment can satisfy all hard formulas in Φ, i.e., there exists no feasible solution.

(*Completeness:*) Assume that there is a completed Weighted Partial Regular MaxSAT tableau T for Φ that does not have cost m. We distinguish two cases:

(i) T has a branch b of cost k, where $k < m$. Then, the total weight of the soft formulas containing the symbol $\square$ in b is k, and there exists a feasible assignment whose total weight of unsatisfied active soft formulas in b is k. By Proposition 5.3.3, this also holds for T_0, but this is in contradiction with m being the minimum total weight of the soft formulas in Φ falsified by any feasible assignment, because $k < m$. Thus, any branch of T has at least cost m.

(ii) T has no branch of cost m. This is in contradiction with the fact that the minimum total weight of the unsatisfied soft formulas in Φ is m. Since the expansion rules preserve the minimum total weight of the unsatisfied soft formulas and the branches not containing the symbol $\blacksquare$ of any completed tableau only contain active soft formulas containing the symbol $\square$ or soft regular literals from which we cannot derive contradictions, T must have a saturated branch where the total weight of the soft formulas containing the symbol $\square$ is m. Thus, T has a branch of cost m.

Hence, each completed tableau T for a multiset Φ of regular formulas has cost m if the minimum total weight of the soft formulas in Φ that are falsified by any feasible assignment is m, and has infinite cost if no such assignment exists.

$\square$

Given a completed Weighted Partial Regular MaxSAT tableau, an optimal assignment is derived by inspecting a branch b with minimum cost. As established in the framework of Weighted Partial Regular MaxSAT, the objective is to identify an assignment that satisfies all hard formulas and minimizes the total weight of unsatisfied soft

formulas. Because every active formula in b is a weighted regular literal that must be satisfied, an optimal assignment v must set each variable x to a truth value from the intersection of the signs of all active weighted regular literals in b that contain x. If no active formula in b contains x, then x can be set to any value of N.

5.4 A Weighted Partial Regular MinSAT Tableau Calculus

This section presents a calculus for Weighted Partial Regular MinSAT. Unlike MaxSAT, where the aim is to minimize the total weight of unsatisfied soft formulas, MinSAT seeks to minimize the total weight of satisfied soft formulas, while satisfying all hard formulas. We use the symbol $\top$ to denote that a soft formula must necessarily be satisfied, and the symbol $\blacksquare$ to denote an infeasible solution, i.e., when no assignment can satisfy all hard formulas.

In the input multiset of formulas Φ, each formula is represented as a pair (ϕ, w), where ϕ is a regular formula and w is its associated weight. We distinguish between two classes of formulas based on their weight: if $w = \infty$, the formula is classified as a hard formula, representing a constraint that cannot be violated. Conversely, if w is a positive integer, it is classified as a soft formula, representing an optimization preference where the objective is to minimize the total weight of satisfied formulas.

The expansion rules of a sound and complete calculus for Weighted Partial Regular MinSAT are shown in Figure 5.5. The calculus works by minimizing the total weight of the satisfied formulas, and the rules replace the premises with the conclusions, rendering the soft premises inactive in the current branch. The cost of a saturated branch is infinite if it contains the symbol $\blacksquare$; otherwise, it is the total weight of the soft formulas containing the symbol $\top$. The cost of a completed tableau is the minimum cost among all its branches. An infinite cost means that there is no feasible solution.

Definition 5.4.1. *A Weighted Partial Regular MinSAT tableau is a rooted tree consisting of a finite number of branches. Each node within this tree is labelled with a hard regular formula, a soft regular formula, the symbol $\blacksquare$, or a formula of the form $(\top, w)$. The symbol $\blacksquare$ denotes an infeasible solution, while $\top$ denotes a satisfied formula, meaning that its weight w must be added to the cost of the branch.*

Definition 5.4.2. *Let $N = \{0, \frac{1}{n-1}, \ldots, \frac{n-1}{n-1}\}$ be a truth value set and let $\Phi = \{A_1, \ldots, A_m\}$ be a multiset of regular formulas. A tableau for Φ is constructed by a sequence of applications of the following rules in such a way that the soft premises of the rule become inactive and the conclusions become active in the branch where the rule is applied:*

Initialize: *We start by generating a tree with a single branch with m nodes such that each node is labelled with a formula of Φ, called the initial tableau. All its formulas are declared to be active.*

$\vee$-rule (soft premise): *Given a tableau T for Φ and a branch b of T, if b contains an active regular formula $(\leq i{:}A \vee B, w)$ (resp. $(\geq i{:}A \vee B, w)$), the tableau obtained*

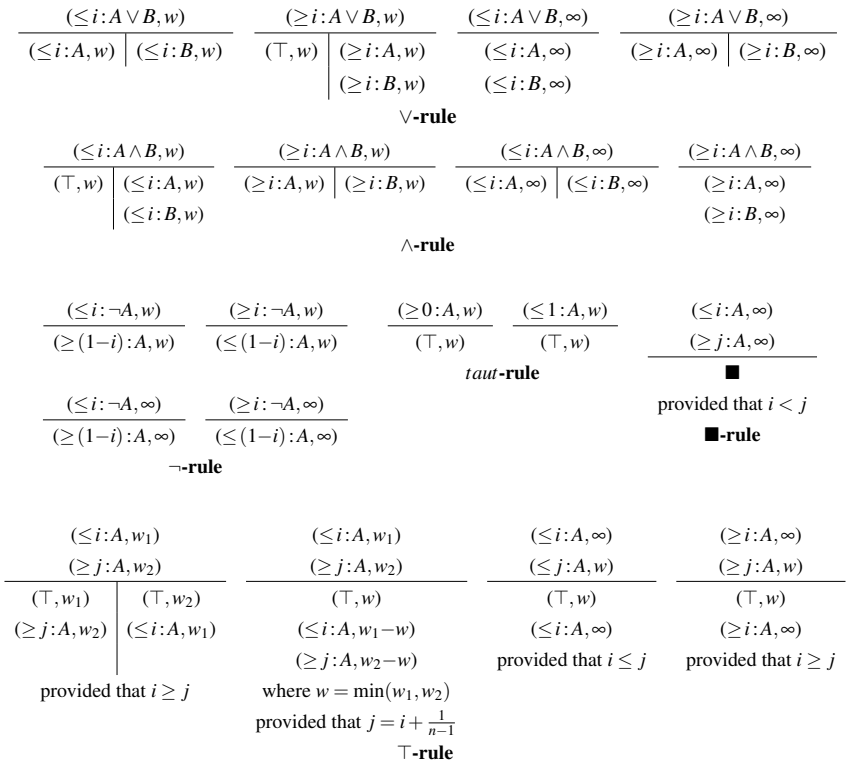


Figure 5.5: Tableau expansion rules for Weighted Partial Regular MinSAT.

by appending a new left branch with a node below b labelled with $(\leq i:A,w)$ (resp. $(\top,w)$) and a new right branch with a node (resp. two nodes) below b labelled with $(\leq i:B,w)$ (resp. $(\geq i:A,w)$ and $(\geq i:B,w)$) is a tableau for Φ.

$\vee$-rule (hard premise): *Given a tableau T for Φ and a branch b of T, if b contains an active regular formula $(\leq i:A\vee B,\infty)$, the tableau obtained by appending a new branch with two nodes below b labelled with $(\leq i:A,\infty)$ and $(\leq i:B,\infty)$ is a tableau for Φ. If b contains an active regular formula $(\geq i:A\vee B,\infty)$, the tableau obtained by appending a new left branch with a node $(\geq i:A,\infty)$ and a new right branch with a node $(\geq i:B,\infty)$ is a tableau for Φ.*

$\wedge$-rule (soft premise): *Given a tableau T for Φ and a branch b of T, if b contains an active regular formula $(\leq i:A\wedge B,w)$ (resp. $(\geq i:A\wedge B,w)$), the tableau obtained by appending a new left branch with a node below b labelled with $(\top,w)$ (resp. $(\geq i:A,w)$) and a new right branch with two nodes (resp. a node) $(\leq i:A,w)$ and $(\leq i:B,w)$ (resp. $(\geq i:B,w)$) is a tableau for Φ.*

$\wedge$-rule (hard premise): *Given a tableau T for Φ and a branch b of T, if b contains an active regular formula $(\leq i:A\wedge B,\infty)$ the tableau obtained by appending a new*

left branch with a node $(\leq i:A,\infty)$ *and a new right branch with a node* $(\leq i:B,\infty)$ *is a tableau for* Φ*. If* b *contains an active regular formula* $(\geq i:A\wedge B,\infty)$*, the tableau obtained by appending a new branch with two nodes below* b *labelled with* $(\geq i:A,\infty)$ *and* $(\geq i:B,\infty)$ *is a tableau for* Φ*.*

$\neg$-rule (soft premise): *If a branch* b *contains an active regular formula* $(\leq i:\neg A,w)$ *(resp.* $(\geq i:\neg A,w)$*), the tableau obtained by appending a node below* b *labelled with* $(\geq (1-i):A,w)$ *(resp.* $(\leq (1-i):A,w)$*) is a tableau for* Φ*.*

$\neg$-rule (hard premise): *If a branch* b *contains an active regular formula* $(\leq i:\neg A,\infty)$ *(resp.* $(\geq i:\neg A,\infty)$*), the tableau obtained by appending a node below* b *labelled with* $(\geq (1-i):A,\infty)$ *(resp.* $(\leq (1-i):A,\infty)$*)) is a tableau for* Φ*.*

*taut***-rule:** *If a branch* b *contains an active regular formula* $(\geq 0:A,w)$*, the tableau obtained by appending a node below* b *labelled with* $(\top,w)$ *is a tableau for* Φ*. If* b *contains an active regular formula* $(\leq 1:A,w)$*, the tableau obtained by appending a node below* b *labelled with* $(\top,w)$ *is a tableau for* Φ*.*

■-rule: *If a branch* b *contains two active regular formulas* $(\leq i:A,\infty)$ *and* $(\geq j:A,\infty)$ *such that* $i<j$*, the tableau obtained by appending a node below* b *labelled with* ■ *is a tableau for* Φ*. In this case, the branch is pruned because this corresponds to an infeasible solution.*

$\top$-rule (two soft premises and $i\geq j$): *If a branch* b *contains two active regular formulas* $(\leq i:A,w_1)$ *and* $(\geq j:A,w_2)$ *such that* $i\geq j$*, the tableau obtained by appending a new left branch with two nodes below* b *labelled with* $(\top,w_1)$ *and* $(\geq j:A,w_2)$*, and a new right branch with two nodes below* b *labelled with* $(\top,w_2)$ *and* $(\leq i:A,w_1)$ *is a tableau for* Φ*.*

$\top$-rule (two soft premises and $j=i+\frac{1}{n-1}$): *If a branch* b *contains two active regular formulas* $(\leq i:A,w_1)$ *and* $(\geq j:A,w_2)$ *such that* $j=i+\frac{1}{n-1}$*, the tableau obtained by appending a new branch with three nodes below* b *labelled with* $(\top,w)$*,* $(\leq i:A,w_1-w)$ *and* $(\geq j:A,w_2-w)$*, where* $w=\min(w_1,w_2)$*, is a tableau for* Φ*.*

$\top$-rule (one soft and one hard premise): *If a branch* b *contains two active regular formulas* $(\leq i:A,\infty)$ *and* $(\leq j:A,w)$ *such that* $i\leq j$*, the tableau obtained by appending a new branch with two nodes below* b *labelled with* $(\top,w)$ *and* $(\leq i:A,\infty)$ *is a tableau for* Φ*. If* b *contains two active regular formulas* $(\geq i:A,\infty)$ *and* $(\geq j:A,w)$ *such that* $i\geq j$*, the tableau obtained by appending a new branch with two nodes below* b *labelled with* $(\top,w)$ *and* $(\geq i:A,\infty)$ *is a tableau for* Φ*.*

Definition 5.4.3. *Let* T *be a tableau for a multiset* Φ *of regular formulas. A branch* b *of* T *is saturated when no further expansion rules can be applied to* b *or the* ■*-rule has pruned it. The tableau* T *is completed when all its branches are saturated. The cost of a saturated branch is the total weight of the formulas containing* $\top$ *in that branch. The cost of a completed tableau is the minimum cost among all its branches.*

Given a multiset Φ of hard and soft regular formulas, we demonstrate below that the minimum total weight of the soft formulas satisfied by any assignment satisfying

all hard formulas is m if and only if the cost of a completed tableau for Φ is m. Thus, the systematic construction of a completed tableau for Φ provides an exact method for solving Weighted Partial Regular MinSAT.

Example 5.4.4. *Figure 5.6 shows how to create a tableau, with the proposed calculus, to minimize the total weight of satisfied soft formulas, while satisfying all hard formulas, in the multiset of weighted regular formulas* $\Phi = \{(\geq 1 : x_1, \infty), (\geq \frac{3}{4} : x_2, \infty), (\leq \frac{3}{4} : x_3, 6), (\geq \frac{1}{2} : x_2 \vee x_3, 3), (\geq \frac{3}{4} : x_1 \wedge x_2, 5)\}$ *, assuming* $N = \{0, \frac{1}{4}, \frac{2}{4}, \frac{3}{4}, 1\}$. *The first tableau (the leftmost tableau on top) is the initial tableau, which contains one node for each regular formula in* Φ. *The second tableau shows the application of the* $\wedge$*-rule to* $(\geq \frac{3}{4} : x_1 \wedge x_2, 5)$*, the application of the* $\top$*-rule to* $(\geq 1 : x_1, \infty)$ *and* $(\geq \frac{3}{4} : x_1, 5)$ *on the leftmost branch, and the application of the* $\top$*-rule to* $(\geq \frac{3}{4} : x_2, 5)$ *and* $(\geq \frac{3}{4} : x_2, \infty)$ *on the rightmost branch. The third tableau shows the application of the* $\vee$*-rule to* $(\geq \frac{1}{2} : x_2 \vee x_3, 3)$. *The fourth tableau shows the application of the* $\top$*-rule to* $(\geq \frac{3}{4} : x_2, \infty)$ *and* $(\geq \frac{1}{2} : x_2, 3)$ *on the second and fourth branches. Finally, the fifth tableau shows the application of the* $\top$*-rule to* $(\leq \frac{3}{4} : x_3, 6)$ *and* $(\geq \frac{1}{2} : x_3, 3)$ *on the second and fourth branches of the previous tableau. Since the tableau is completed and the minimum cost among its branches is 8, the minimum total weight of soft formulas in* Φ *that are satisfied by any assignment satisfying all hard formulas is also 8.*

5.4.1 Soundness and completeness

We prove the soundness and completeness of the proposed calculus. Before that, we prove termination and the soundness of the expansion rules.

Proposition 5.4.5. *A Weighted Partial Regular MinSAT tableau for a multiset* Φ *of regular weighted formulas is completed in a finite number of steps.*

Proof. We first create an initial tableau and then apply expansion rules in the newly created branches until they become saturated. The $\vee$-, $\wedge$- and $\neg$-rules reduce the number of connectives. Since the number of connectives in the initial formula is finite and no rule introduces new connectives, these rules can only be applied a finite number of times. The $\top$-rule has two regular formulas as premises and each conclusion contains exactly one formula in the conclusion that could be a premise of another rule because a formula containing $\top$ cannot be a premise. Note that the second $\top$-rule has two conclusions that can be a premise, $(\leq i : A, w_1 - w)$ and $(\geq j : A, w_2 - w)$, but at most one of them is added because the weight of at least one of the conclusions is 0. Therefore, the $\top$-rule can only be applied a finite number of times. Regarding the *taut*-rule, the conclusion is of the form $(\top, w)$, which cannot be a premise. Regarding the $\blacksquare$-rule, its application prunes the branch. Hence, the construction of any completed tableau terminates in a finite number of steps. □

It suffices to establish the soundness for the functionally complete set $\{\neg, \vee\}$, since every other connective is definable in terms of these two: $A \rightarrow B$ is equivalent to $\neg A \vee B$, and $A \wedge B$ is equivalent to $\neg(\neg A \vee \neg B)$. The soundness of the $\wedge$-rule then follows from the soundness of the $\neg$-rule and $\vee$-rule. Consequently, soundness is proved below for $\vee$, $\neg$, *taut*, $\blacksquare$ and $\top$, with the $\wedge$-rule understood to follow by an analogous argument.

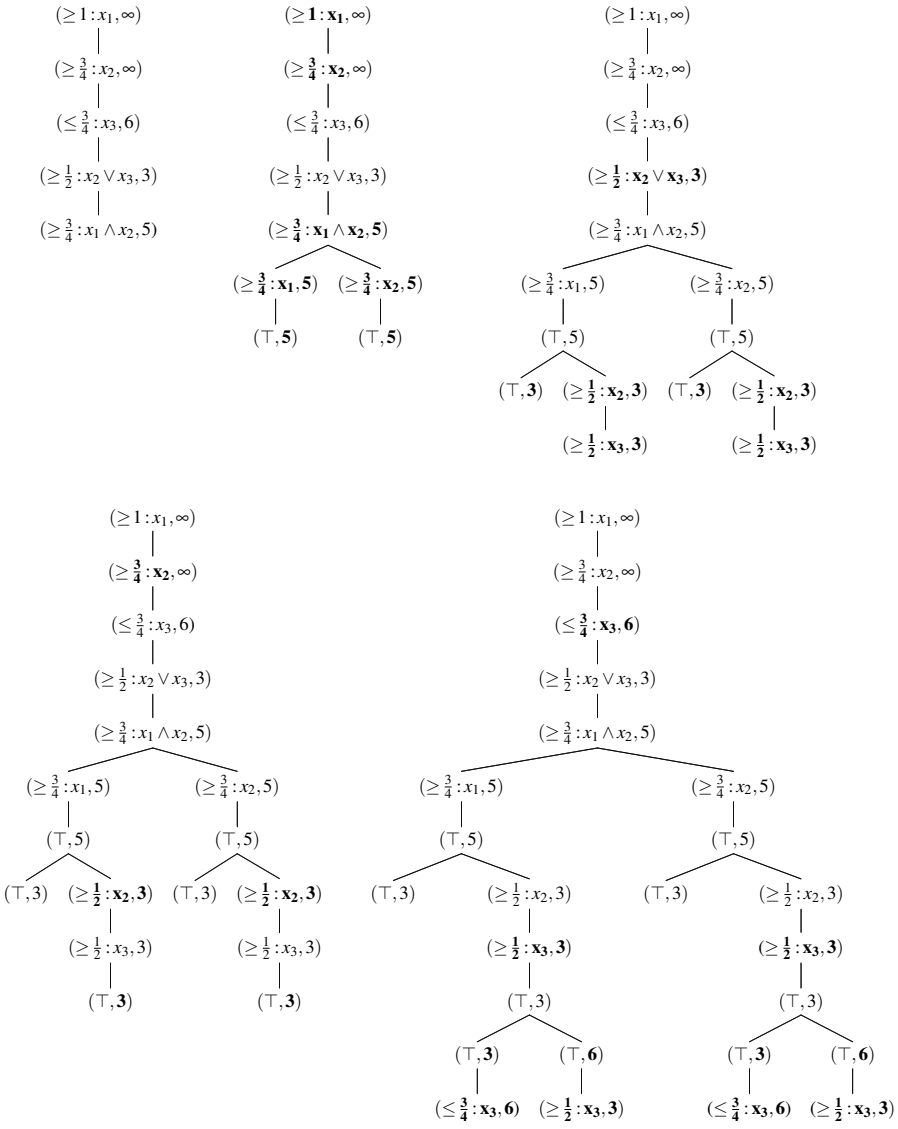


Figure 5.6: A completed tableau for the Weighted Partial Regular MinSAT instance $\{(\geq 1 : x_1, \infty), (\geq \frac{3}{4} : x_2, \infty), (\leq \frac{3}{4} : x_3, 6), (\geq \frac{1}{2} : x_2 \vee x_3, 3), (\geq \frac{3}{4} : x_1 \wedge x_2, 5)\}$, assuming $N = \{0, \frac{1}{4}, \frac{2}{4}, \frac{3}{4}, 1\}$.

Proposition 5.4.6. *The expansion rules of the tableau calculus for Weighted Partial Regular MinSAT in Figure 5.5 are sound in the following sense: for any assignment v, the total weight of the satisfied soft premises is preserved in at least one branch and does not decrease in the remaining branches. Furthermore, rules with hard premises preserve the satisfiability from the premises to the conclusions in at least one branch.*

Proof. We demonstrate the soundness of each rule separately, assuming that the truth value set is $N = \{0, \frac{1}{n-1}, \ldots, \frac{n-1}{n-1}\}$:

- $\vee$**-rule (sign** $\leq$**):** If v satisfies $(\leq i : A \vee B, w)$, it holds that $v(A) \leq i$ and $v(B) \leq i$, because of the maximum function. Hence, v satisfies all the conclusions, which contain exactly one formula and the weight is preserved. If v does not satisfy $(\leq i : A \vee B, w)$, it holds that $v(A) > i$ or $v(B) > i$. Hence, v unsatisfies at least one conclusion and the minimum weight of satisfied formulas in the conclusions is 0. The weights are preserved because the premise and the conclusion contain a single formula.

  When the premise is $(\leq i : A \vee B, \infty)$, satisfiability is preserved because v satisfies $(\leq i : A \vee B, \infty)$ if and only if $v(A) \leq i$ and $v(B) \leq i$.Thus, the premise is satisfied if and only if all the conclusions are satisfied.

- $\vee$**-rule (sign** $\geq$**):** If v satisfies $(\geq i : A \vee B, w)$, it holds that $v(A) \geq i$ or $v(B) \geq i$. Hence, there is one satisfied formula on the left branch and at least one satisfied formula on the right branch and, therefore, the minimum weight is w. If v does not satisfy $(\geq i : A \vee B, w)$, it holds that $v(A) < i$ and $v(B) < i$, and the minimum weight is 0 because the two formulas in the right conclusion are unsatisfied. A violated hard regular formula prunes the branch.

  When the premise is $(\geq i : A \vee B, \infty)$, satisfiability is preserved because v satisfies $(\geq i : A \vee B, \infty)$ if and only if $v(A) \geq i$ or $v(B) \geq i$. Thus, the premise is satisfied if and only if at least one conclusion is satisfied.

- $\neg$**-rule:** The soundness of this rule when the premise contains a formula of the form $(\leq i : \neg A, w)$ follows from the fact that the single conclusion is satisfied if and only if the premise is satisfied, because an assignment satisfies $\leq i : \neg A$ if and only if $v(\neg A) = 1 - v(A) \leq i$ (i.e.; $v(A) \geq 1 - i$). Similar arguments apply when the premise contains a formula of the form $(\geq i : \neg A, w)$ and when the formula is hard.

- *taut*-**rule:** Since the premise is a tautology, the single conclusion must necessarily be satisfied and preserve the weight.

- $\blacksquare$**-rule:** If two regular hard formulas cannot be simultaneously satisfied, there is no feasible solution.

- $\top$**-rule (with two weighted soft premises):** If $i \geq j$, at least one premise is satisfied because the intersection of the signs of the premises is non-empty. If $v(A) > i$, exactly one premise and one formula in the right conclusion are satisfied, and two formulas in the left conclusion are satisfied. If $j \leq v(A) \leq i$, two premises and two formulas in each conclusion are satisfied. If $v(A) < j$, exactly

one premise and one formula in the left conclusion are satisfied, and two formulas in the right conclusion are satisfied. Therefore, the total weight of the satisfied soft formulas is preserved between the premises and the conclusions. If $j = i + \frac{1}{n-1}$, we have that the intersection of the signs is empty and their union is the whole truth value set N. Therefore, the total weight of the satisfied soft formulas is preserved because there is always one satisfied and one unsatisfied premise.

- $\top$**-rule (with one hard premise and one weighted soft premise):** We must ensure the satisfaction of the hard premise. Given that these two rules are only applicable when the sign of the soft premise contains the sign of the hard premise, the soft premise must necessarily be satisfied. Therefore, the total weight of the satisfied soft formulas is preserved between the premises and the conclusions.

□

Given a completed tableau T for a multiset Φ of hard and soft regular formulas, the soundness and completeness theorem establishes that the cost of T is m if and only if the minimum total weight of the soft formulas in Φ that are satisfied by any feasible assignment is m.

Theorem 5.4.7. Soundness & completeness. *Let T be a completed Weighted Partial Regular MinSAT tableau for a multiset Φ of weighted regular formulas. Then, the cost of T is m if and only if the minimum total weight of the soft formulas in Φ that are satisfied by any assignment satisfying all hard formulas is m; and the cost of T is infinite if and only if no assignment satisfies all hard formulas.*

Proof. (*Soundness:*) T was derived by creating a sequence of tableaux $T_0, \ldots, T_n$ such that T_0 is an initial tableau for Φ, $T_n = T$, and T_i was obtained by a single application of one of the expansion rules in Figure 5.5 on a branch of T_{i-1}, for $i = 1, \ldots, n$, starting with an initial tableau that only contains the formulas in Φ. By Proposition 5.4.5, we know that such a sequence is finite. Since T has cost m, T_n contains a branch b that does not contain the symbol ■ such that the total weight of the formulas containing the symbol $\top$ is m, and the total weight of the formulas containing $\top$ in the rest of the branches that do not contain ■ is at least m. Moreover, the active soft formulas in every branch of T_n that does not contain the symbol ■ are weighted regular literals of the form $(\leq i : x, w)$ or $(\geq j : x, w')$ such that, for each propositional variable x, they can be unsatisfied; otherwise, we could yet apply expansion rules and T_n would not be completed. Notice that, in a saturated branch that does not contain the symbol ■, the union of the regular signs, for each propositional variable x, is a strict subset of the truth value set.

Given a saturated branch b with cost m, let i_h be the smallest i in the active hard formulas of the form $(\leq i : x, \infty)$ and let j_h be the greatest j in the active hard formulas of the form $(\geq j : x, \infty)$. If there is no hard formula with negative (resp. positive) polarity, then $i_h = 1$ (resp. $j_h = 0$). To ensure that v satisfies the hard formulas, we must consider an assignment v that sets each variable x to one value k such that $j_h \leq k \leq i_h$. Indeed, x must be assigned a value in $N_h = \{j_h, j_h + \frac{1}{n-1}, \ldots, i_h - \frac{1}{n-1}, i_h\}$. On the other hand, let i_s be the greatest i in the active soft formulas of the form $(\leq i : x, w)$ and j_s be

the smallest j in the active soft formulas of the form $(\geq j : x, w)$. If there is no soft formula with negative (resp. positive) polarity, then $i_s = i_h$ (resp. $j_s = j_h$). We consider $N_s = \{i_s, i_s + \frac{1}{n-1}, \ldots, j_s - \frac{1}{n-1}, j_s\}$. By the application of the expansion rules, if b has active soft formulas containing variable x, it holds that $N_s \subset N_h$; otherwise, $N_s = N_h$. To ensure that v satisfies the hard formulas and the active formulas containing the symbol $\top$ but does not satisfy the rest of the active soft formulas, k must be in $N_h \setminus N_s$. If no soft formula must be unsatisfied, k can be any value of N_h. Moreover, there cannot be any assignment where the total weight of the satisfied soft formulas is less than m in a branch of T_n, because the formulas containing $\top$ are always satisfied. Therefore, the minimum total weight of the active soft formulas that are satisfied by any feasible assignment among the branches of T_n is m.

(*Completeness:*) Assume that there is a completed Weighted Partial Regular MinSAT tableau T for Φ that does not have cost m. We distinguish two cases:

(i) T has a branch b of cost k, where $k < m$. Then, the total weight of the soft formulas containing the symbol $\top$ in b is k, and there exists a feasible assignment whose total weight of satisfied active soft formulas in b is k. By Proposition 5.4.6, this also holds for T_0, but this is in contradiction with m being the minimum total weight of the soft formulas in Φ satisfied by any feasible assignment because $k < m$. Thus, any branch of T has at least cost m.

(ii) T has no branch of cost m. This is in contradiction with the fact that the minimum total weight of the satisfied soft formulas in Φ is m. Since the expansion rules preserve the minimum total weight of the satisfied soft formulas and the branches not containing the symbol $\blacksquare$ of any completed tableau only contain active soft formulas containing the symbol $\top$ or soft regular literals that can be unsatisfied, T must have a saturated branch where the total weight of the soft formulas containing the symbol $\top$ is m. Thus, T has a branch of cost m.

Hence, each completed tableau T for a multiset Φ of regular formulas has cost m if the minimum total weight of the soft formulas in Φ that are satisfied by any feasible assignment is m, and has infinite cost if no such assignment exists. □

Observe that the proof of soundness provides a method for building an optimal assignment from a saturated branch of minimal cost. Because the expansion rules may be applied in any order, the cost of the resulting completed tableau remains invariant, as Proposition 5.4.6 ensures that the minimum total weight of the satisfied soft formulas is preserved at every step.

5.5 Clausal Tableau Calculi

The proposed tableau calculi for Regular MaxSAT and Regular MinSAT operate in a non-clausal setting, handling multisets of arbitrary regular formulas built from propositional variables and the connectives $\{\vee, \wedge, \neg, \rightarrow\}$. This section introduces the clausal variants of these calculi, which restrict the input to multisets of regular clauses, and analyzes the formal relationship between both approaches.

Definition 5.5.1. *A regular clause is a regular formula of the form* $\geq i : l_1 \vee l_2 \vee \cdots \vee l_n$

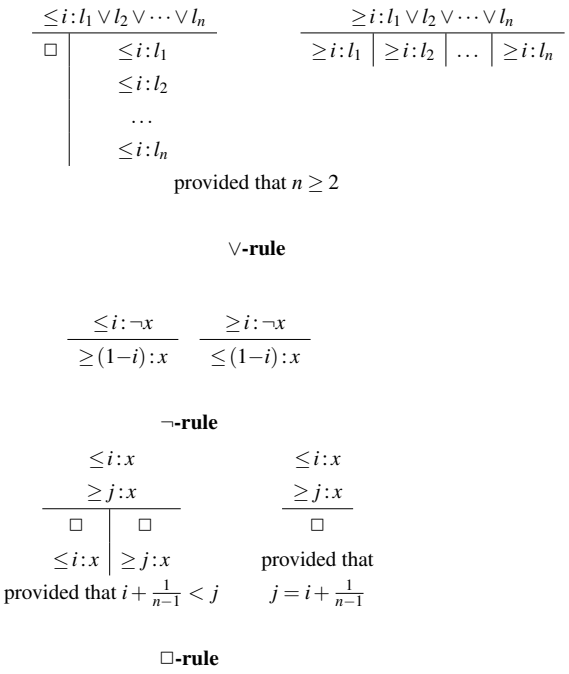


Figure 5.7: Tableau expansion rules for Clausal Regular MaxSAT.

or $\leq i : l_1 \vee l_2 \vee \cdots \vee l_n$*, where each* l_j *is a propositional variable* x *or a negated propositional variable* $\neg x$*.*

The restriction to clausal form has a direct consequence on the structure of the calculus. Since every formula in the input multiset is by definition a disjunction of literals, the connectives $\wedge$ and $\rightarrow$ do not appear, and their corresponding expansion rules are not required. The clausal calculus, therefore, consists solely of the $\vee$-rule, the $\neg$-rule and the $\Box$-rule.

Figure 5.7 displays the expansion rules for a clausal calculus for Regular MaxSAT. Observe that the premises of both the $\neg$-rule and the $\Box$-rule are restricted to regular literals rather than to arbitrary regular formulas. Furthermore, since we are only dealing with disjunctions of literals, the clausal version of the $\vee$-rule collapses $n-1$ applications of the non-clausal $\vee$-rule.

The notions of branch saturation, tableau completion, and branch cost are identical to those in the non-clausal calculus. Furthermore, the soundness and completeness of the clausal calculus are directly inherited from the non-clausal version.

The clausal calculus in Figure 5.7 extends to the weighted and partial settings: The weights are propagated from the premises to the conclusions in an analogous manner, and the distinction between hard and soft formulas is handled identically. Figure 5.8 shows a complete clausal calculus for Weighted Partial Regular MaxSAT. Given that the underlying theoretical principles governing MaxSAT remain equally valid in the context of MinSAT, we straightforwardly present the expansion rules for a complete

clausal calculus for Weighted Partial Regular MinSAT in Figure 5.9.

For the sake of conciseness, formal proofs are omitted herein as they closely parallel those established for the non-clausal setting. A comprehensive exposition and the proofs of soundness and completeness can be found in [35].

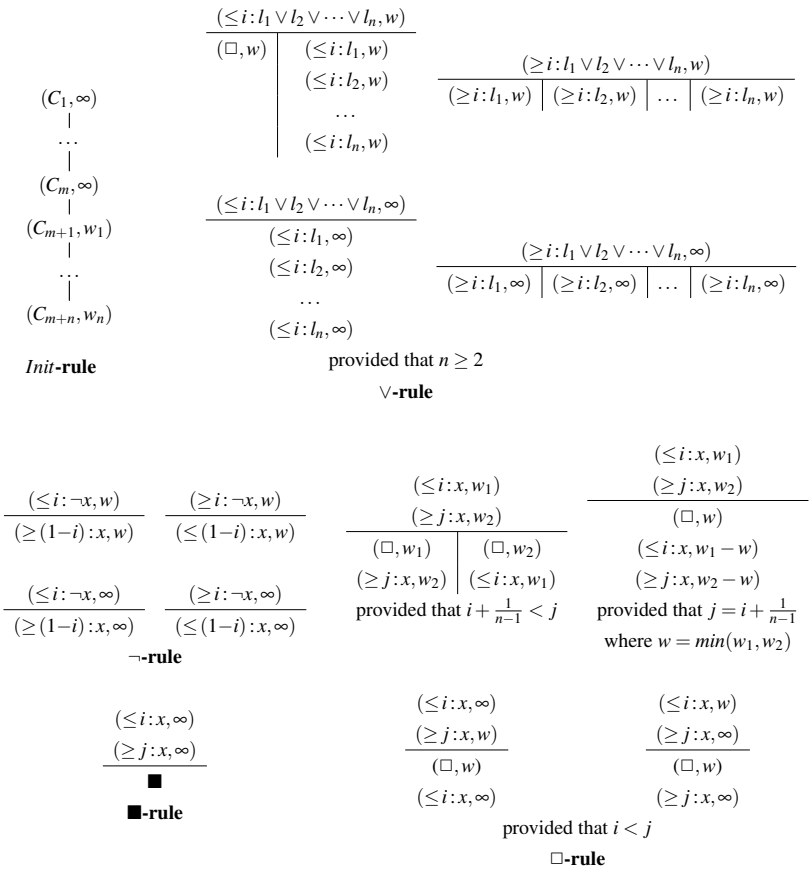


Figure 5.8: A tableau calculus for Clausal Weighted Partial Regular MaxSAT.

5.6 Solving Regular MaxSAT and Regular MinSAT with MILP

Although the previous tableau calculi provide an exact method for Regular MaxSAT and MinSAT, their direct implementation is not competitive in practice. Nevertheless, their inference rules strongly inspired the Mixed-Integer Linear Programming (MILP) reduction introduced in this section, which reformulates the same reasoning principles within a more efficient optimization framework.

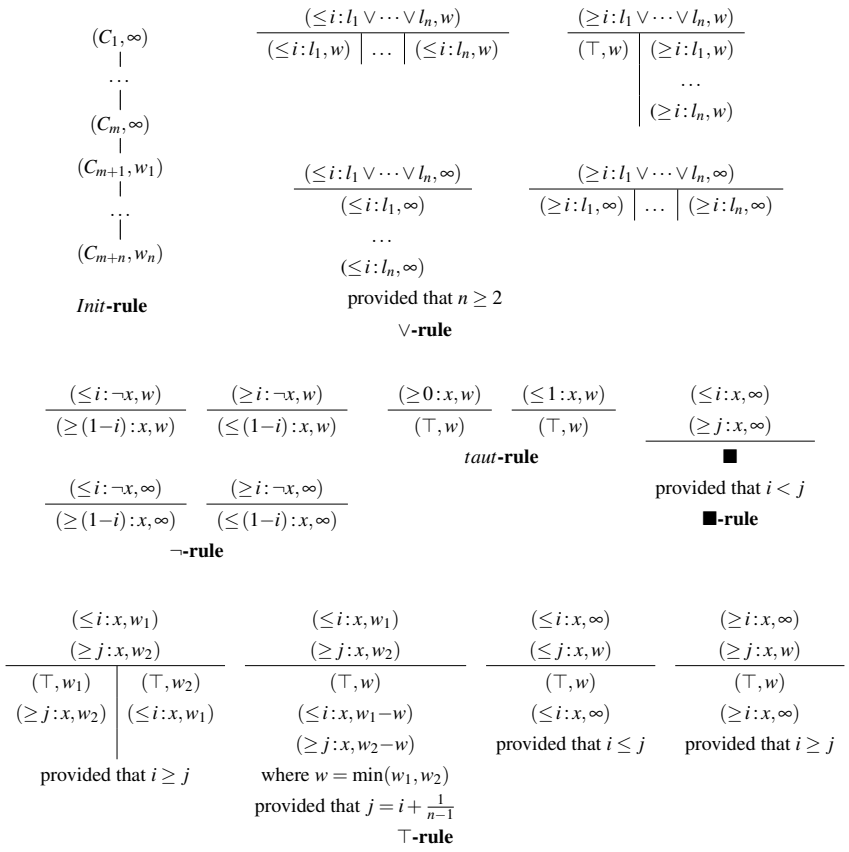


Figure 5.9: A tableau calculus for Clausal Weighted Partial Regular MinSAT.

Following the approach developed to solve SAT [50, 43] and MaxSAT [51] in Łukasiewicz logic, we derive an MILP encoding for Regular MaxSAT and delegate the task of finding an optimal solution to a general-purpose MILP solver. This yields an exact method for Regular MaxSAT that benefits from decades of engineering advances in branch-and-bound algorithms, cutting-plane techniques, and linear-programming relaxations.

Since the distinction between MaxSAT and MinSAT in this context only affects the cost function, we focus our presentation on Regular MaxSAT and defer Regular MinSAT to the end of this section.

5.6.1 An MILP Encoding for Regular MaxSAT

Given a multiset of regular formulas $\Phi=\{S_1:A_1,\ldots,S_m:A_m\}$ evaluated over a truth value set N, our aim is to derive an MILP encoding to solve the Regular MaxSAT problem for Φ using an MILP solver. The proposed MILP formulation considers three types of variables to capture the essential elements of Regular MaxSAT:

- Definition variables: For each subformula B of A_i, where $S_i : A_i \in \Phi$, we introduce a variable $z_B \in N$ that denotes the truth value assigned to B. The formula A_i has its corresponding variable z_{A_i}.
- Violation variables: For each regular formula $S_i : A_i \in \Phi$, we introduce a binary variable $p_i \in \{0,1\}$. In an optimal assignment, p_i is set to 0 if $S_i : A_i$ is satisfied; otherwise, it is violated.
- Auxiliary variables: Fresh binary variables $c, c', c'', \ldots$ are needed to encode the interpretation of the connectives as linear equations.

In addition to establishing the variables, modeling a linear optimization problem involves defining both the objective function and the constraints that define the feasible region. The objective function is

$$\min \sum_{i=1}^{m} p_i,$$

where each p_i is a violation variable. Minimizing $\sum_{i=1}^{m} p_i$ enforces the satisfiability of $S_i : A_i$ in an optimal solution when p_i evaluates to 0. In this way, the number of violation variables set to 0 in an optimal solution is the maximum number of regular formulas in Φthat are satisfied, and the optimal value returned by the objective function is the minimum number of regular formulas that are falsified by any assignment.

The constraints that define the feasible region consist of a set of linear expressions designed to ensure that there exists an interpretation that satisfies formula $S_i : A_i$ when p_i is 0. When p_i cannot be set to 0 in an optimal solution, p_i enforces the satisfaction of the constraints, but $S_i : A_i$ cannot be satisfied.

We consider three types of constraints: Given a regular formula $\geq i : A$ ($\leq i : A$), the first type define the values that variables can take, the second type compute the truth value that an interpretation assigns to A, and the third type checks whether this value is greater (smaller) than or equal to i. We refer to them as domain constraints, semantic constraints, and regular constraints, respectively.

Domain Constraints

The proposed MILP encoding is valid for both finite and infinite truth value sets. In the infinite case, the truth value set is assumed to be the interval $N = [0,1]$. In the finite case, it is assumed to be the set of equidistant rational numbers between 0 and 1 inclusive, defined as $N = \{0, \frac{1}{n-1}, \frac{2}{n-1}, \ldots, 1\}$.

Domain constraints establish the boundaries or the discrete grid for each definition variable z. When the truth-value set N is the continuous interval $[0,1]$, each z is simply declared as a continuous variable bounded by $[0,1]$. However, handling the finite case requires additional constraints, as standard MILP solvers natively support only continuous intervals and integer domains.

A finite set of rational values, such as $N = \{0, \frac{1}{2}, 1\}$ for a three-valued logic, cannot be declared directly. To overcome this limitation, we introduce binary indicator variables $b_{xk} \in \{0,1\}$, associated with each grid point $v_k \in N$. Since the variables are

binary and constrained to sum to one, exactly one truth value is selected. Formally, this restriction is modeled by the following equations:

$$\sum_{k=1}^{n} b_{xk} = 1 \tag{5.1}$$

$$z_x = \sum_{k=1}^{n} v_k \cdot b_{xk} \tag{5.2}$$

For example, consider a three-valued logic where $N = \{0, \frac{1}{2}, 1\}$. In this case, each variable x is assigned three binary indicators: $[b_{x1}, b_{x2}, b_{x3}]$. The first equation enforces that $b_{x1} + b_{x2} + b_{x3} = 1$, restricting the feasible assignments of the vector (b_{x1}, b_{x2}, b_{x3}) to $(1,0,0)$, $(0,1,0)$, or $(0,0,1)$. Then, by the second equation, variable z_x can only take values from the discrete set $\{0, \frac{1}{2}, 1\}$.

Regular Constraints

The regular constraint for a regular formula $\geq i : A_i$ ($\leq i : A_i$) checks whether the value assigned to A_i, represented by the assignment to variable z_{A_i}, is greater (smaller) than or equal to i.

Given a multiset of regular formulas $\Phi = \{S_1 : A_1, \ldots, S_m : A_m\}$, we add the following regular constraints for each regular formula $S_k : A_k \in \Phi$:

- If $S_k : A_k = \geq i_k : A_k$, we add the linear inequality:

$$z_{A_k} + p_k \geq i_k \tag{5.3}$$

- If $S_k : A_k = \leq i_k : A_k$, we add the linear inequality:

$$z_{A_k} - p_k \leq i_k \tag{5.4}$$

Observe that, when $p_k = 0$, the first inequality becomes $z_{A_k} \geq i_k$, and the second inequality becomes $z_{A_k} \leq i_k$. Since z_{A_k} denotes the value assigned to A_k, it follows that $S_k : A_k$ is satisfied. In contrast, when $p_k = 1$, both inequalities are trivially satisfied for any values of z_{A_k} and i_k given that $0 \leq z_{A_k}, i_k \leq 1$. This implies that all regular constraints are satisfied by any feasible solution, but $S_k : A_k$ is not necessarily satisfied. Additionally, by maximizing the number of violation variables assigned the value 0, one ensures that $S_k : A_k$ is violated in an optimal solution whenever $p_k = 1$.

These transformations can be represented by the following inference rules, which replace the premise by its conclusions:

$$\frac{\geq i : A}{\begin{array}{c} z_A + p \geq i \\ z_A = A \end{array}} \geq\text{-rule} \qquad \frac{\leq i : A}{\begin{array}{c} z_A - p \leq i \\ z_A = A \end{array}} \leq\text{-rule}$$

where p is a binary variable, called a violation variable.

Semantic Constraints

In the MILP encoding, logical connectives are interpreted by translating their definitions into systems of linear constraints. For each subformula of a given formula, we recursively introduce a definition variable to capture its truth value and apply the inference rule associated with its dominant connective. These inference rules, which replace the premise by its conclusions, are applied recursively until no further rules can be applied. In the end, we get a set of linear constraints.

The rules for conjunction, disjunction, and negation are defined as follows:

$$\frac{z_{A\wedge B} = A \wedge B}{\begin{array}{c} z_{A\wedge B} \leq z_A \\ z_{A\wedge B} \leq z_B \\ z_{A\wedge B} \geq z_A - c \\ z_{A\wedge B} \geq z_B - (1-c) \\ z_A = A \\ z_B = B \end{array}} \wedge\text{-rule} \qquad \frac{z_{A\vee B} = A \vee B}{\begin{array}{c} z_{A\vee B} \geq z_A \\ z_{A\vee B} \geq z_B \\ z_{A\vee B} \leq z_A + c \\ z_{A\vee B} \leq z_B + (1-c) \\ z_A = A \\ z_B = B \end{array}} \vee\text{-rule} \qquad \frac{z_{\neg A} = \neg A}{\begin{array}{c} z_{\neg A} = 1 - z_A \\ z_A = A \end{array}} \neg\text{-rule}$$

where c is a fresh auxiliary variable.

The $\wedge$-rule encodes $z_{A\wedge B} = \min(z_A, z_B)$, the $\vee$-rule encodes $z_{A\vee B} = \max(z_A, z_B)$, the $\neg$-rule encodes $z_{\neg A} = 1 - z_A$. Notice that, at the end of the rules, we add $z_A = A$ and $z_B = B$ so that we must recursively apply the rules if A or B are not propositional variables.

We illustrate this MILP encoding with a concrete example below and prove the correctness of the previous inference rules in the next section.

Example 5.6.1. *Let $N = \{0, 0.1, \ldots, 0.9, 1\}$ be a truth value set and let $\Phi = \{A_1, A_2, A_3\}$ be a multiset of regular formulas, where $A_1 = \geq 0.2 : x_1 \wedge (x_2 \vee \neg x_3)$, $A_2 = \leq 0.1 : x_1$, and $A_3 = \geq 0.5 : \neg x_2 \vee x_3$. We illustrate the proposed encoding by reducing the Regular MaxSAT instance Φ to an MILP instance.*

The objective function is

$$\min \sum_{i=1}^{3} p_i,$$

and is subject to the following constraints:

$$z_{x_1 \wedge (x_2 \vee \neg x_3)} + p_1 \geq 0.2 \tag{5.5}$$

$$z_{x_1} - p_2 \leq 0.1 \tag{5.6}$$

$$z_{\neg x_2 \vee x_3} + p_3 \geq 0.5 \tag{5.7}$$

$$\begin{aligned} z_{x_1 \wedge (x_2 \vee \neg x_3)} &= x_1 \wedge (x_2 \vee \neg x_3) \\ z_{x_1} &= x_1 \\ z_{\neg x_2 \vee x_3} &= \neg x_2 \vee x_3 \end{aligned} \tag{5.8}$$

In order to derive an MILP instance, we must translate the last three equations into linear constraints. Notice that the second equation, $z_{x_1} = x_1$, is already linear and therefore no transformation is required. For the rest of the equations, we must apply the tableau-based rules defined in Section 5.6.1.

We start by transforming $z_{x_1 \wedge (x_2 \vee \neg x_3)} = x_1 \wedge (x_2 \vee \neg x_3)$. First, we apply the $\wedge$-rule:

$$z_{x_1 \wedge (x_2 \vee \neg x_3)} \leq z_{x_1} \tag{5.9}$$
$$z_{x_1 \wedge (x_2 \vee \neg x_3)} \leq z_{x_2 \vee \neg x_3} \tag{5.10}$$
$$z_{x_1 \wedge (x_2 \vee \neg x_3)} \geq z_{x_1} - c \tag{5.11}$$
$$z_{x_1 \wedge (x_2 \vee \neg x_3)} \geq z_{x_2 \vee \neg x_3} - (1 - c) \tag{5.12}$$
$$z_{x_1} = x_1 \tag{5.13}$$
$$z_{x_2 \vee \neg x_3} = x_2 \vee \neg x_3$$

Since $z_{x_2 \vee \neg x_3} = x_2 \vee \neg x_3$ is not a linear constraint, we apply the $\vee$-rule to it:

$$z_{x_2 \vee \neg x_3} \geq z_{x_2} \tag{5.14}$$
$$z_{x_2 \vee \neg x_3} \geq z_{\neg x_3} \tag{5.15}$$
$$z_{x_2 \vee \neg x_3} \leq z_{x_2} + c' \tag{5.16}$$
$$z_{x_2 \vee \neg x_3} \leq z_{\neg x_3} + (1 - c') \tag{5.17}$$
$$z_{x_2} = x_2 \tag{5.18}$$
$$z_{\neg x_3} = \neg x_3$$

Since $z_{\neg x_3} = \neg x_3$ is not a linear constraint, we apply the $\neg$-rule to it:

$$z_{\neg x_3} = 1 - z_{x_3} \tag{5.19}$$
$$z_{x_3} = x_3 \tag{5.20}$$

We now transform $z_{\neg x_2 \vee x_3} = \neg x_2 \vee x_3$. We first apply the $\vee$-rule to it:

$$z_{\neg x_2 \vee x_3} \geq z_{\neg x_2} \tag{5.21}$$
$$z_{\neg x_2 \vee x_3} \geq z_{x_3} \tag{5.22}$$
$$z_{\neg x_2 \vee x_3} \leq z_{\neg x_2} + c'' \tag{5.23}$$
$$z_{\neg x_2 \vee x_3} \leq z_{x_3} + (1 - c'') \tag{5.24}$$
$$z_{\neg x_2} = \neg x_2$$
$$z_{x_3} = x_3 \tag{5.25}$$

Since $z_{\neg x_2} = \neg x_2$ is not a linear constraint, we apply the $\neg$-rule to it:

$$z_{\neg x_2} = 1 - z_{x_2} \tag{5.26}$$
$$z_{x_2} = x_2 \tag{5.27}$$

The MILP encoding of Φ consists of the objective function along with the linear constraints 5.5–5.27 and the corresponding domain constraints. Solving this MILP formulation with a state-of-the-art solver such as HiGHS yields an optimal solution to the Regular MaxSAT instance Φ.

Correctness of the Transformation

Lemma 5.6.2. *Assume $z_A, z_B, z_{A\wedge B} \in N$. Then, $z_{A\wedge B} = \min(z_A, z_B)$ if and only if there exists $c \in \{0,1\}$ such that the following constraints hold:*

$$z_{A\wedge B} \leq z_A \tag{5.28}$$
$$z_{A\wedge B} \leq z_B \tag{5.29}$$
$$z_{A\wedge B} \geq z_A - c \tag{5.30}$$
$$z_{A\wedge B} \geq z_B - (1-c) \tag{5.31}$$

Proof. Assume $z_\phi = z_{A\wedge B} = \min(z_A, z_B)$.

Case 1: Suppose $z_\phi = z_A$. Constraint 5.28 and Constraint 5.29 hold since $z_\phi = \min(z_A, z_B)$. Setting $c = 0$, Constraint 5.30 becomes $z_\phi \geq z_A$, which holds since $z_\phi = z_A$, and Constraint 5.31 becomes $z_\phi \geq z_B - 1$, which holds because $z_B - 1 \leq 0$.

Case 2: Suppose $z_\phi = z_B$. Constraint 5.28 and Constraint 5.29 hold since $z_\phi = \min(z_A, z_B)$. Setting $c = 1$, Constraint 5.30 becomes $z_\phi \geq z_A - 1$, which holds because $z_A - 1 \leq 0$, and Constraint 5.31 becomes $z_\phi \geq z_B$, which holds since $z_\phi = z_B$.

Now, assume there exists $c \in \{0,1\}$ such that Constraints 5.28,5.29,5.30 and 5.31 hold.

Case 1: If $c = 0$, then Constraint 5.30 becomes $z_\phi \geq z_A$ and $z_\phi = z_A$ by Constraint 5.28. Since $z_B \geq z_A$ by Constraint 5.29, it holds $z_\phi = \min(z_A, z_B)$.

Case 2: If $c = 1$, then Constraint 5.31 becomes $z_\phi \geq z_B$ and $z_\phi = z_B$ by Constraint 5.29. Since $z_A \geq z_B$ by Constraint 5.28, it holds $z_\phi = \min(z_A, z_B)$. □

Lemma 5.6.3. *Assume $z_A, z_B, z_{A\vee B} \in N$. Then, $z_{A\vee B} = \max(z_A, z_B)$ if and only if there exists $c \in \{0,1\}$ such that all the following constraints hold:*

$$z_{A\vee B} \geq z_A \tag{5.32}$$
$$z_{A\vee B} \geq z_B \tag{5.33}$$
$$z_{A\vee B} \leq z_A + c \tag{5.34}$$
$$z_{A\vee B} \leq z_B + (1-c) \tag{5.35}$$

Proof. Assume $z_\phi = z_{A\vee B} = \max(z_A, z_B)$.

Case 1: Suppose $z_\phi = z_A$. Constraint 5.32 and Constraint 5.33 hold since $z_\phi = \max(z_A, z_B)$. Setting $c = 0$, Constraint 5.34 becomes $z_\phi \leq z_A$, which holds since $z_\phi = z_A$, and Constraint 5.35 becomes $z_\phi \leq z_B + 1$, which holds because $z_B + 1 \geq 1$.

Case 2: Suppose $z_\phi = z_B$. Constraint 5.32 and Constraint 5.33 hold since $z_\phi = \max(z_A, z_B)$. Setting $c = 1$, Constraint 5.34 becomes $z_\phi \leq z_A + 1$, which holds because $z_A + 1 \geq 1$, and Constraint 5.35 becomes $z_\phi \leq z_B$, which holds since $z_\phi = z_B$.

Now, assume there exists $c \in \{0,1\}$ such that Constraints 5.32,5.33, 5.34, and 5.35 hold.

Case 1: If $c = 0$, then Constraint 5.34 becomes $z_\phi \leq z_A$ and $z_\phi = z_A$ by Constraint 5.32. Since $z_A \geq z_B$ by Constraint 5.33, it holds $z_\phi = \max(z_A, z_B)$.

Case 2: If $c = 1$, then Constraint 5.35 becomes $z_\phi \leq z_B$ and $z_\phi = z_B$ by Constraint 5.33. Since $z_B \geq z_A$ by Constraint 5.32, it holds $z_\phi = \max(z_A, z_B)$. □

Lemma 5.6.4. *Assume* $z_\phi, z_{\neg\phi} \in N$. *Then,* $z_{\neg\phi} = 1 - z_\phi$.

Proof. This follows from the interpretation of the negation connective. □

Note that the number of rule applications to a regular formula is finite, as the input regular formulas contain only finitely many connectives and each rule application eliminates exactly one connective.

5.6.2 An MILP encoding for Weighted Partial Regular MaxSAT

In Weighted Partial Regular MaxSAT, the problem formulation includes both hard and soft formulas. Hard formulas are satisfied under all circumstances, while soft formulas are associated with a weight that quantifies the penalty incurred when they are violated.

We first consider Weighted Regular MaxSAT. In this case, we are given a multiset of soft formulas $\Phi = \{(S_1 : A_1, w_1), \ldots, (S_m : A_m, w_m)\}$, and the goal is to minimize the total weight of the unsatisfied formulas. The MILP encoding is similar to that in the unweighted case, with the key difference being the objective function, which now is

$$\min \sum_{i=1}^{m} w_i \times p_i.$$

The regular constraint of $(\geq i_k : A_k, w_k)$ is still $z_{A_k} + p_k \geq i_k$, and the regular constraint of $(\leq i_k : A_k, w_k)$ is $z_{A_k} - p_k \leq i_k$. Finally, the semantic constraints are also derived from the expression $z_{A_k} = A_k$ by applying the transformation rules described in Section 5.6.1.

The previous transformations can be represented by the following inference rules, which replace the premise by its conclusions:

$$\frac{(\geq i : A, w)}{\begin{array}{c} z_A + p \geq i \\ z_A = A \end{array}} \geq\text{-rule} \qquad \frac{(\leq i : A, w)}{\begin{array}{c} z_A - p \leq i \\ z_A = A \end{array}} \leq\text{-rule}$$

where p is a fresh violation variable.

In Weighted Partial Regular MaxSAT, where we also have hard formulas, we must take into account that they are satisfied by any assignment and, therefore, are not part of the objective function. To this end, we need to add to the rules defined in Section 5.6.1 the following rules:

$$\frac{(\geq i : A \wedge B, \infty)}{\begin{array}{c} z_{A\wedge B} \geq i \\ z_{A\wedge B} = A \wedge B \end{array}} \qquad \frac{(\leq i : A \wedge B, \infty)}{\begin{array}{c} z_{A\wedge B} \leq i \\ z_{A\wedge B} = A \wedge B \end{array}}$$

$\wedge$-rule-pos $\wedge$-rule-neg

$$\frac{(\geq i : A \vee B, \infty)}{\begin{array}{c} z_{A\vee B} \geq i \\ z_{A\vee B} = A \vee B \end{array}} \qquad \frac{(\leq i : A \vee B, \infty)}{\begin{array}{c} z_{A\vee B} \leq i \\ z_{A\vee B} = A \vee B \end{array}}$$

$\vee$-rule-pos $\vee$-rule-neg

$$\frac{(\geq i : \neg A, \infty)}{\begin{array}{c} z_A \leq 1 - i \\ z_A = A \end{array}} \qquad \frac{(\leq i : \neg A, \infty)}{\begin{array}{c} z_A \geq 1 - i \\ z_A = A \end{array}}$$

$\neg$-rule-pos $\neg$-rule-neg

The MILP formulation is added to the constraints of the feasible region as before, but violation variables are omitted in the regular constraints of hard regular formulas.

Example 5.6.5. *Let $N = [0,1]$ be a truth value set and let $\Phi = \{A_1, A_2, A_3\}$ be a Weighted Partial Regular MaxSAT instance, where $A_1 = (\geq 0.5 : x_1 \wedge x_2, \infty)$ is a hard formula, and $A_2 = (\leq 0 : x_1, 3)$ and $A_3 = (\leq 0 : x_2, 2)$ are soft formulas. We illustrate the proposed approach by reducing the Weighted Partial Regular MaxSAT instance Φ to an MILP instance.*

The objective function is $\min(3p_2 + 2p_3)$*, and is subject to the following constraints:*

$$z_{x_1 \wedge x_2} \geq 0.5 \tag{5.36}$$
$$z_{x_1 \wedge x_2} = x_1 \wedge x_2$$
$$z_{x_1 \wedge x_2} \leq z_{x_1} \tag{5.37}$$
$$z_{x_1 \wedge x_2} \leq z_{x_2} \tag{5.38}$$
$$z_{x_1 \wedge x_2} \geq z_{x_1} - c \tag{5.39}$$
$$z_{x_1 \wedge x_2} \geq z_{x_2} - (1 - c) \tag{5.40}$$
$$z_{x_1} = x_1 \tag{5.41}$$
$$z_{x_2} = x_2 \tag{5.42}$$
$$z_{x_1} - p_2 \leq 0 \tag{5.43}$$
$$z_{x_2} - p_3 \leq 0 \tag{5.44}$$

The MILP encoding of Φ consists of the objective function along with the linear constraints 5.36–5.44 and the corresponding domain constraints.

5.6.3 An MILP Encoding for Weighted Partial Regular MinSAT

Weighted Partial Regular MinSAT instances consist of both hard and soft formulas. Hard formulas must always be satisfied, whereas soft formulas are associated with a weight representing the cost incurred when they are satisfied.

Given a multiset of regular formulas

$$\Phi = \{(S_1 : A_1, w_1), \ldots, (S_k : A_k, w_k), (S_{k+1} : A_{k+1}, \infty), \ldots, (S_m : A_m, \infty)\},$$

where the first k formulas are soft and the remainder are hard, the MILP encoding for Weighted Partial Regular MinSAT is identical to that of Weighted Partial Regular MaxSAT, except for the objective function, which becomes

$$\min \sum_{i=1}^{k} w_i \cdot (1 - p_i) \tag{5.45}$$

Recall that, in our encoding, $p_i = 0$ indicates that the corresponding regular formula is satisfied. Therefore, the term $1 - p_i$ evaluates to 1 exactly when the formula is satisfied, ensuring that the associated weight w_i contributes to the objective function.

5.6.4 Experimental Evaluation

This section presents the results of an empirical evaluation of the proposed MILP-based approach on randomly generated Max-2SAT and Max-3SAT instances.

We generated random Max-2SAT and Max-3SAT instances over two kinds of truth value sets: the continuous interval $[0,1]$ and the discrete sets $N = \{0, \frac{1}{n-1}, \ldots, 1\}$ for $n \in \{3,5,7\}$. For each truth value set, the clause-to-variable ratio ranged from 1 to 50 in steps of 1, and we generated 50 instances for each ratio. Each Max-2SAT instance consists of clauses of the form $\leq i : \ell_1 \vee \ell_2$ or $\geq i : \ell_1 \vee \ell_2$, and each Max-3SAT instance consists of clauses of the form $\leq i : \ell_1 \vee \ell_2 \vee \ell_3$ or $\geq i : \ell_1 \vee \ell_2 \vee \ell_3$. The literals of each clause are sampled uniformly without replacement and negated with 0.5 probability. The regular sign of each clause is chosen uniformly at random among all regular signs allowed by the fixed truth value set. All these instances were translated into MILP programs using the encoding defined in Section 5.6 and then solved with HiGHS [54], an open-source exact MILP solver, using a time limit of 1800 seconds. We accessed HiGHS via SciPy, a Python scientific computing library. SciPy does not solve the problem itself, but it builds the model and passes it to HiGHS.

We record two measures against the ratio α. The first is the satisfaction rate, which indicates the mean percentage of satisfied clauses. The second is the solving time, computed for each ratio as the mean value of the 50 instances.

All experiments were performed in the Ars Magna high-performance computing cluster at the Artificial Intelligence Research Institute (IIIA-CSIC). Each node is equipped with two Intel Xeon Silver 4210 CPUs at 2.2 GHz (10 cores per CPU) and 92 GB of RAM. We ran each Regular MaxSAT instance as a separate SLURM array task.

Figures 5.10 and 5.11 report the satisfaction rate for Max-2SAT and Max-3SAT, respectively. In Max-2SAT, the satisfaction rate starts around 0.9 at $\alpha = 1$ for every

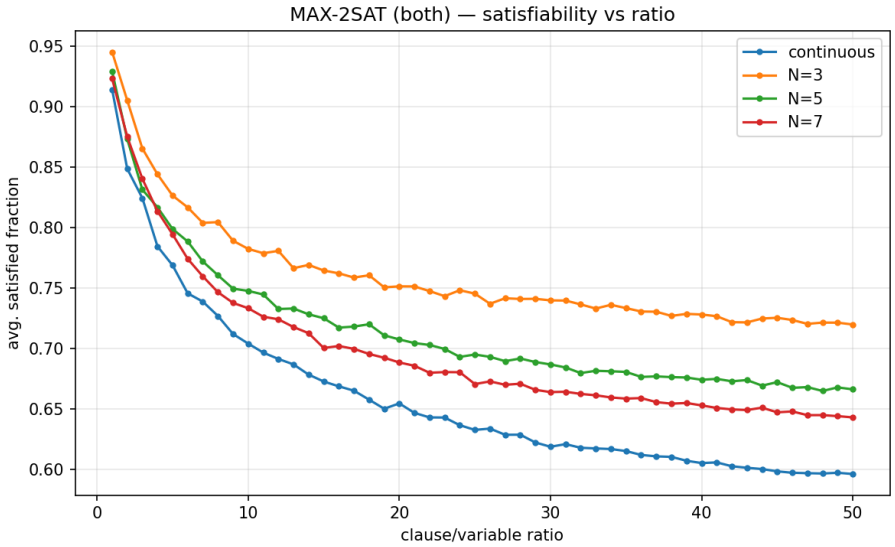


Figure 5.10: Regular Max-2SAT: Satisfaction rate against the clause-to-variable ratio α, one curve per truth-value set.

truth value set, decreases rapidly as the ratio α increases, and eventually levels off. The relative order of the truth value sets is preserved over the whole range, and the finite sets remain above the continuous case. The smallest set, $N = 3$, yields the highest satisfaction rate and levels off around 0.72, whereas the continuous case decreases to about 0.60. Max-3SAT exhibits the same ordering, but the satisfaction rate is lower for every truth value set, and the continuous case decreases to about 0.56. Overall, the satisfaction rate decreases as the cardinality of the truth value set increases, and the continuous case yields the lowest rate.

Figures 5.12 and 5.13 report the solving time for Max-2SAT and Max-3SAT, respectively. In Max-2SAT, the solving time remains close to zero for small values of α. As α increases, the solving time rises steeply and then flattens at the 1800 second time limit. The ratio at which this happens depends on the truth value set. The continuous case reaches the limit first, around $\alpha = 20$, closely followed by $N = 7$ and then $N = 5$, whereas $N = 3$ reaches it only past $\alpha = 40$. Max-3SAT follows the same pattern, but every truth value set reaches the limit at a lower ratio, and the continuous case flattens already around $\alpha = 12$. The reason is that the three-literal encoding produces larger MILP programs. Overall, the solving time increases with the cardinality of the truth value set.

The experimental results show that the proposed MILP reduction provides a practical exact method for Regular Max-2SAT and Max-3SAT over both finite and continuous truth value sets, and merits further investigation as an algorithmic component integrated into the broader repertoire of exact approaches to many-valued MaxSAT and MinSAT.

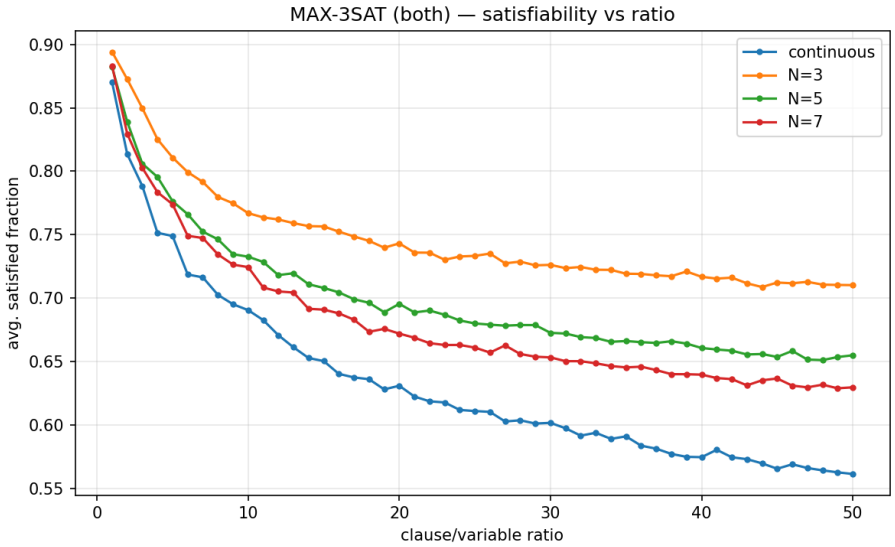


Figure 5.11: Regular Max-3SAT: Satisfaction rate against the clause-to-variable ratio α, one curve per truth-value set.

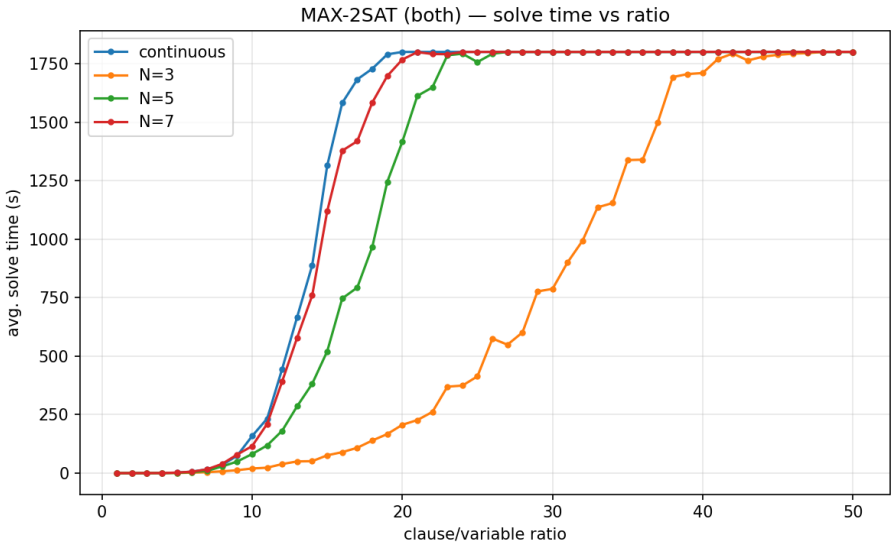


Figure 5.12: Regular Max-2SAT solving time against the clause-to-variable ratio α, one curve per truth-value set. The time limit is 1800 seconds.

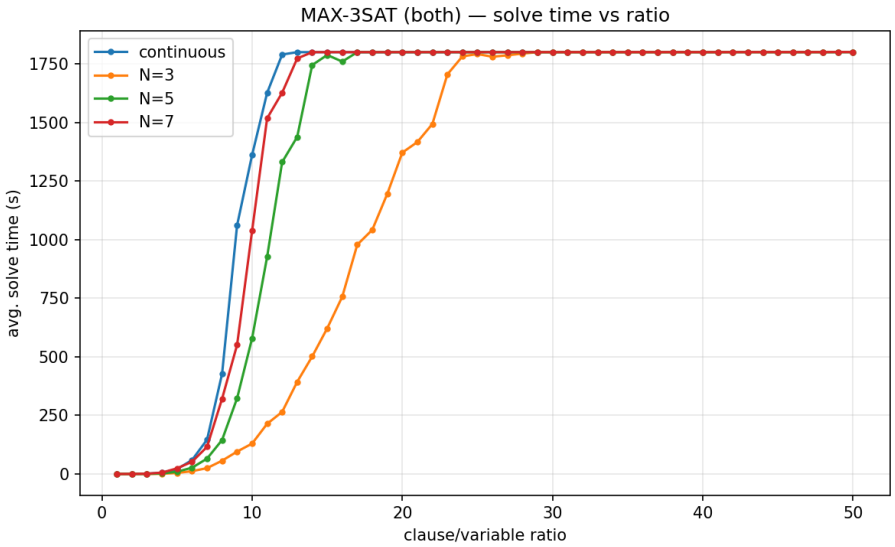


Figure 5.13: Regular Max-3SAT solving time against the clause-to-variable ratio α, one curve per truth-value set. The time limit is 1800 seconds.

5.7 Conclusions

We have developed tableau calculi for Weighted Partial Regular MaxSAT and Weighted Partial Regular MinSAT in both clausal and non-clausal frameworks, and proved their soundness and completeness. In addition, we have presented a reduction of Regular MaxSAT to Mixed Integer Linear Programming (MILP) and assessed its effectiveness through experiments on randomly generated instances.

Chapter 6

Tableaux for Signed MaxSAT and Signed MinSAT

This chapter introduces novel tableau calculi for solving the MaxSAT and MinSAT problems within the framework of signed propositional logic, establishing their soundness and completeness for both Weighted Partial Signed MaxSAT and Weighted Partial Signed MinSAT. Since their expansion rules depend on the cardinality of the truth value set and lack uniformity, these calculi operate over a fixed three-valued domain.

6.1 Introduction

Signed logic generalizes Boolean logic by considering a finite set N of truth values and associating each formula with a sign $S \subseteq N$, which represents a subset of possible truth values. Within this framework, an assignment is a mapping that assigns a truth value from N to each propositional variable, which is extended to arbitrary formulas by interpreting conjunction, disjunction, and negation through the minimum, maximum, and Łukasiewicz negation functions, respectively [11, 49]. An assignment v satisfies a signed formula $S:A$ if and only if $v(A) \in S$. Given a multiset of such formulas, the MaxSAT (MinSAT) problem aims to find an assignment that maximizes (minimizes) the number of satisfied signed formulas [3, 4, 2].

The unrestricted nature of arbitrary signs increases the complexity of proof systems, as detecting contradictions can require more than two signed literals and computing intersections is less efficient. More importantly, unlike in regular logic, this lack of sign structure prevents the definition of tableau proof systems that remain uniform across arbitrary domain sizes; instead, the number of branches in the expansion rules grows with the cardinality of N, thereby increasing the search space complexity for the corresponding tableau solvers. Driven by this limitation, in this chapter, we fix the truth value set to $N = \{0, \frac{1}{2}, 1\}$ and introduce novel tableau calculi for both Weighted Partial Signed MaxSAT and Weighted Partial Signed MinSAT, establishing their soundness and completeness.

The remainder of this chapter is structured as follows. Section 6.2 introduces the Signed MaxSAT tableau calculus. Section 6.3 extends it to the weighted partial setting and establishes its soundness and completeness. Section 6.4 presents the Signed MinSAT tableau calculus. Section 6.5 extends it to the weighted partial setting and proves its soundness and completeness. Finally, Section 6.6 concludes the chapter.

6.2 A Signed MaxSAT Tableau Calculus

We introduce a sound and complete tableau calculus for Signed MaxSAT, assuming $N = \{0, \frac{1}{2}, 1\}$. We address Signed MaxSAT by determining the minimum number of formulas that are falsified by any assignment. Figure 6.1 displays the expansion rules of this calculus. As in all inference rules for MaxSAT, we substitute premises with conclusions.

The rules are defined so that the number of unsatisfied formulas in the premise is maintained in at least one branch and does not decrease in the remaining branches. As a result of applying these rules, formulas within a branch are categorized as active or inactive: active formulas can function as premises, whereas inactive ones cannot. Finally, since the unweighted case is subsumed by the weighted case, we only prove the soundness and completeness for Weighted Partial Signed MaxSAT (cf. Theorem 6.3.4).

Definition 6.2.1. *A Signed MaxSAT tableau T for a multiset Φ of signed formulas is a tree with a finite number of branches whose nodes are labelled by a signed formula or the symbol $\square$, which denotes a contradiction. A branch is a path from the root to a leaf, and we assume that branches have a finite number of nodes.*

Definition 6.2.2. *A tableau T for Φ is constructed by a sequence of applications of the rules in Figure 6.1. We start by constructing the initial tableau, which consists of a single branch where each node is labelled with a formula in Φ and all nodes are marked as active in this branch. An expansion rule can be applied to a branch b if and only if all premises of the rule are active in b. After applying the rule, branch b is split into a number $n \geq 1$ of branches $b_1, \ldots, b_n$. Each new branch marks the premises as inactive and introduces a number of new active nodes, as defined in Figure 6.1.*

Definition 6.2.3. *Let T be a Signed MaxSAT tableau for a multiset Φ of signed formulas. A branch b of T is saturated when no further expansion rules can be applied on b, and T is completed when all its branches are saturated. The cost of a saturated branch is the number of occurrences of the symbol $\square$ in the branch. The cost of a completed tableau is the minimum cost among all its branches.*

Given a completed tableau T for a multiset Φ of signed formulas, the soundness and completeness theorem establishes that the cost of T is m if and only if the minimum number of signed formulas in Φ that are falsified by any assignment is m.

Theorem 6.2.4. Soundness & completeness. *Let T be a completed Signed MaxSAT tableau for a multiset Φ of signed formulas. Then, the cost of T is m if and only if the minimum number of signed formulas in Φ that are falsified by any assignment is m.*

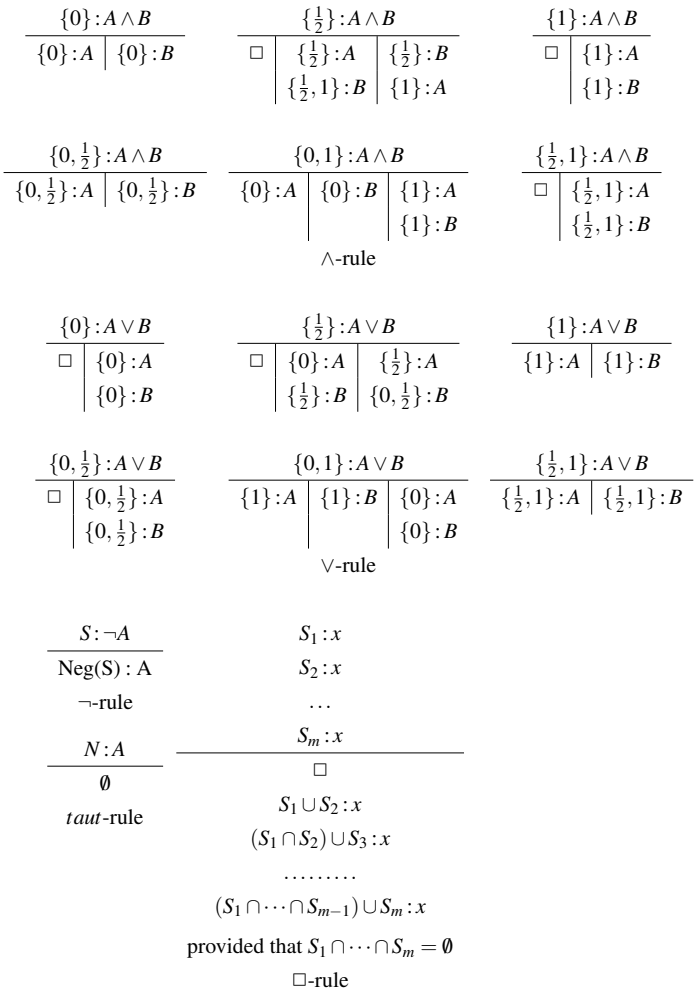


Figure 6.1: A complete tableau calculus for Signed MaxSAT.

Example 6.2.5. *Figure 6.2 shows the steps needed to create a completed Signed MaxSAT tableau for the multiset* Φ=$\{\{1\}:x_1\wedge x_2,\{0,\frac{1}{2}\}:x_1\vee x_2\}$*, assuming* $N=\{0,\frac{1}{2},1\}$*. The first tableau (the leftmost tableau on top) is the initial tableau, which contains one node for each formula in* Φ*. The second tableau applies the* $\wedge$*-rule to* $\{1\}:x_1\wedge x_2$*. The third tableau applies the* $\vee$*-rule to* $\{0,\frac{1}{2}\}:x_1\vee x_2$*. The fourth tableau applies the* $\square$*-rule to* $\{0,\frac{1}{2}\}:x_1$ *and* $\{1\}:x_1$*, and to* $\{0,\frac{1}{2}\}:x_2$ *and* $\{1\}:x_2$*. Notice that we do not add* $\{0,\frac{1}{2},1\}:x_1$ *and* $\{0,\frac{1}{2},1\}:x_2$ *in the rightmost branch because they can be removed by the taut-rule. Since the tableau is completed and the minimum cost among its branches is 1, the minimum number of formulas in* Φ *that are falsified by any assignment is also 1. This claim follows from Theorem 6.2.4.*

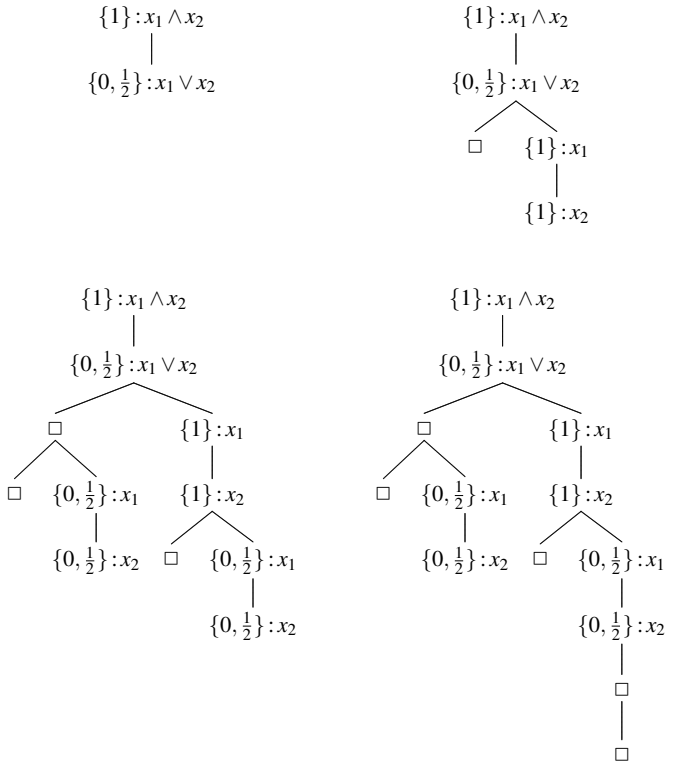


Figure 6.2: A completed Signed MaxSAT tableau for the multiset of signed formulas $\{\{1\}:x_1 \wedge x_2, \{0,\frac{1}{2}\}:x_1 \vee x_2\}$, where $N=\{0,\frac{1}{2},1\}$.

6.3 A Weighted Partial Signed MaxSAT Tableau Calculus

The expansion rules for the proposed complete tableau calculus for Weighted Partial Signed MaxSAT are detailed in Figure 6.3. For the $\wedge$-, $\vee$-, and $\neg$-rules, a distinction is made between expansion rules with a soft premise and those with a hard premise. When there is exactly one soft premise, the expansion rules are identical to those in Figure 6.1, except for the fact that they propagate the weight from the premise to the conclusions. Observe that the satisfaction of a formula does not depend on its weight, which is the penalty to be paid if the formula is not satisfied. Conceptually, the application of a soft rule with weight w can be understood as the application of the corresponding unweighted rule w times.

Rules that have hard premises are designed to preserve satisfiability rather than the total weight of unsatisfied soft formulas. That is, any assignment satisfying all premises must satisfy all hard formulas in at least one branch, and any assignment falsifying any

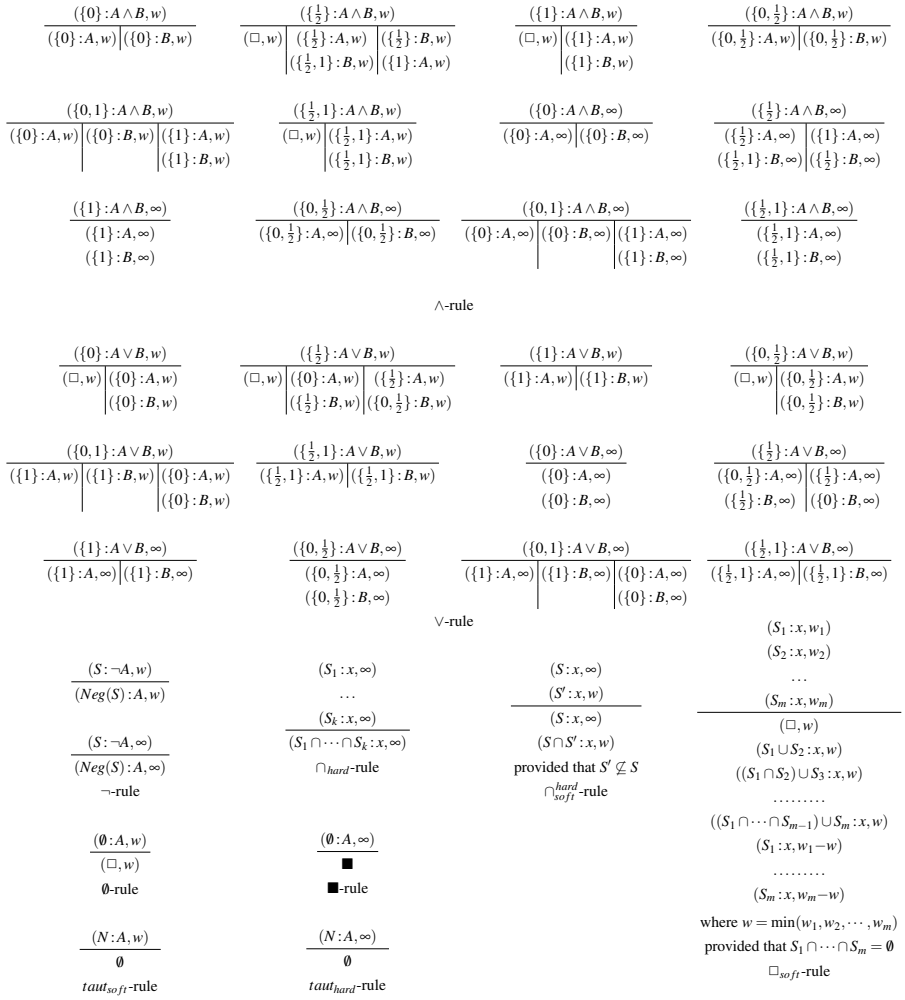


Figure 6.3: A complete tableau calculus for Weighted Partial Signed MaxSAT.

premise must falsify at least one hard formula in each branch. This approach aligns with the rules of a signed SAT calculus [49].

When the hard part becomes inconsistent, the symbol ■ is derived, which means that the branch corresponds to an infeasible solution and, therefore, is pruned. The ■- and $\cap_{hard}$-rules are used to detect infeasible solutions. The $\cap_{hard}$-rule combines multiple hard literals into a single one. This simplification is essential to discard assignments that would result in infeasible solutions.

The ∅-rule states that, from a soft premise with an empty sign and weight w, a contradiction with the same weight can be derived. The $taut_{soft}$- and $taut_{hard}$-rules remove tautologies.

The $\cap^{hard}_{soft}$-rule applies to a hard literal $(S:x,\infty)$ and a soft literal $(S':x,w)$. This

rule restricts the possible values of the soft premise to ensure the satisfaction of the hard premise, allowing the derivation of a contradiction when $S \cap S' = \emptyset$.

The $\square_{soft}$-rule is an adaptation of the unweighted $\square$-rule to the weighted setting. It propagates the minimum weight among the weights of the premises and introduces compensation signed literals to ensure consistency among the weights. The minimum weight is used because the premises may involve literals with different weights. The aim of this rule is to detect when a set of soft literals is inconsistent.

In Weighted Partial Signed MaxSAT, the cost of a saturated branch is infinite if it contains the symbol $\blacksquare$; otherwise, it is the total weight of the soft signed formulas containing the symbol $\square$. The cost of a completed tableau is the minimum cost among all its branches. An infinite cost means that there is no feasible solution.

6.3.1 Soundness and Completeness

Given a completed tableau T for a multiset Φ of hard and soft signed formulas, the soundness and completeness theorem establishes that the cost of T is m if and only if the minimum total weight of the soft formulas in Φ that are falsified by any assignment satisfying all hard formulas is m. Before proving the theorem, we establish termination and the soundness of the expansion rules.

Proposition 6.3.1. *A Weighted Partial Signed MaxSAT tableau for a multiset Φ of weighted signed formulas is completed in a finite number of steps.*

Proof. We first create an initial tableau for Φ and then apply expansion rules in the newly created branches until they become saturated. The $\wedge$-, $\vee$-, and $\neg$-rules reduce the number of connectives. Since the number of connectives in the initial formula is finite and no rule introduces new connectives, these rules can only be applied a finite number of times. The $taut_{soft}$- and $taut_{hard}$-rules remove tautological literals and their conclusions are empty. The $\emptyset$-rule replaces a soft literal with an empty sign by $(\square, w)$, which cannot serve as a premise of any rule. The $\blacksquare$-rule detects infeasibility in the hard part and prunes the branch, after which no further rules apply. The $\cap_{hard}$-rule strictly reduces the number of hard literals by merging them into a single literal with a smaller or equal sign; it can therefore only be applied finitely many times. The $\cap^{hard}_{soft}$-rule applies only when $S' \not\subseteq S$, strictly shrinking the sign of the soft premise, which ensures termination. The $\square_{soft}$-rule strictly reduces the total weight of signed literals containing variable x. Since weights are positive numbers, this process must end. Hence, the construction of any completed tableau terminates in a finite number of steps. $\square$

Proposition 6.3.2. *Given the multisets of weighted signed literals $\{(S_1 : x, w_1), (S_2 : x, w_2)\}$ and $\{((S_1 \cap S_2) : x, w), ((S_1 \cup S_2) : x, w), (S_1 : x, w_1 - w), (S_2 : x, w_2 - w)\}$, where $w = \min(w_1, w_2)$, any assignment v assigns the same total weight to the unsatisfied formulas in both multisets.*

Proof. We distinguish four cases according to whether v satisfies each literal.

- v satisfies $S_1 : x$ and $S_2 : x$: Since $v(x) \in S_1$ and $v(x) \in S_2$, v satisfies $(S_1 \cap S_2) : x$, $(S_1 \cup S_2) : x$, $S_1 : x$ and $S_2 : x$. The total weight of the unsatisfied formulas is 0 in both multisets.

- v satisfies $S_1 : x$ but does not satisfy $S_2 : x$: Since $v(x) \in S_1$ and $v(x) \notin S_2$, v does not satisfy $(S_1 \cap S_2) : x$ but satisfies $(S_1 \cup S_2) : x$. The compensation literal $S_1 : x$ is satisfied and $S_2 : x$ is not. The total weight of the unsatisfied formulas is w_2 in the first multiset and is $w + (w_2 - w) = w_2$ in the second multiset. Both totals agree.
- v does not satisfy $S_1 : x$ but satisfies $S_2 : x$: By symmetry with the previous case, the total weight of the unsatisfied formulas is w_1 in both multisets.
- v satisfies neither $S_1 : x$ nor $S_2 : x$: Since $v(x) \notin S_1$ and $v(x) \notin S_2$, v satisfies neither $(S_1 \cap S_2) : x$ nor $(S_1 \cup S_2) : x$ and falsifies both compensation literals. The total weight of unsatisfied formulas is $w_1 + w_2$ in both multisets.

□

Proposition 6.3.3. *The expansion rules of the Weighted Partial Signed MaxSAT calculus in Figure 6.3 are sound in the sense that the total weight of unsatisfied formulas in the premises is preserved in at least one branch and does not decrease in the remaining branches when the premises are soft. Besides, these rules preserve the satisfiability between the premises and the conclusions in at least one branch when the premises are hard.*

Proof. To prove the soundness of the $\wedge$- and $\vee$-rules with a single soft premise of weight w, we must demonstrate that if an assignment v falsifies the premise, there is one branch in the conclusions where v falsifies one formula, and v falsifies at least one formula in the rest of the branches. Besides, if v satisfies the premise, v satisfies all formulas in at least one branch. Since all cases are similar, we only prove it for $(\{0\} : A \vee B, w)$. In this case, an assignment v falsifies $(\{0\} : A \vee B, w)$ if and only if:

- $v(A) \neq 0$: In the left branch, v falsifies $(\square, w)$; in the right branch, v falsifies at least one formula, which is $(\{0\} : A, w)$.
- $v(B) \neq 0$: In the left branch, v falsifies $\square$; in the right branch, v falsifies at least one formula, which is $(\{0\} : B, w)$.

Moreover, $(\{0\} : A \vee B, w)$ is satisfied when $v(A) = 0$ and $v(B) = 0$. In this case, all the formulas in the right branch are satisfied. Hence, the $\vee$-rule for $(\{0\} : A \vee B, w)$ preserves the total weight of unsatisfied soft formulas between the premises and the conclusions.

The $\neg$-rule propagates the weight unchanged and preserves satisfaction by the semantics of negation. The $taut_{soft}$-rule removes a formula that is always satisfied, which does not affect the total weight of unsatisfied soft formulas.

For rules with a *hard* premise, the goal is to preserve satisfiability rather than the total weight of unsatisfied formulas. The $\wedge$- and $\vee$-rules with a hard premise are the ones used in Signed SAT calculi and preserve satisfiability [49]: any assignment satisfying the hard premise satisfies all conclusions in at least one branch, and any assignment violating it violates some conclusion in every branch.

The $taut_{hard}$-rule removes a tautology, which is always satisfied.

The ■-rule applies when there exist two hard literals $(S_1 : x, \infty)$ and $(S_2 : x, \infty)$ such that $S_1 \cap S_2 = \emptyset$, meaning that no assignment can satisfy both simultaneously, so the branch is declared infeasible.

The $\cap_{hard}$-rule replaces a set of hard literals $(S_1 : x, \infty), \ldots, (S_k : x, \infty)$ with the single literal $((S_1 \cap \cdots \cap S_k) : x, \infty)$. An assignment satisfies all original hard literals if and only if $v(x) \in S_1 \cap \cdots \cap S_k$, so satisfiability is preserved.

The $\cap^{hard}_{soft}$-rule applies when $S' \not\subseteq S$ and replaces the soft literal $(S' : x, w)$ with $((S \cap S') : x, w)$. Any assignment satisfying the hard literal $(S : x, \infty)$ has $v(x) \in S$, so it satisfies the original soft literal $(S' : x, w)$ if and only if $v(x) \in S \cap S'$. Thus, the rule preserves both the satisfiability of the hard formula and the weight of the soft formula.

The $\emptyset$-rule derives $(\square, w)$ from a soft literal with an empty sign. Since no assignment can satisfy a literal with an empty sign, the premise and the conclusion contribute with weight w.

The $\square_{soft}$-rule with m soft premises is proved by applying Proposition 6.3.2 a total of $m - 1$ times. First, Proposition 6.3.2 is applied to $(S_1 : x, w_1)$ and $(S_2 : x, w_2)$, replacing them with $((S_1 \cup S_2) : x, w)$, $((S_1 \cap S_2) : x, w)$, $(S_1 : x, w_1 - w)$, and $(S_2 : x, w_2 - w)$, where $w = \min(w_1, \ldots, w_m)$. In the i-th application, Proposition 6.3.2 is applied to $((S_1 \cap \cdots \cap S_{i-1}) : x, w)$ and $(S_i : x, w_i)$, replacing them with $(((S_1 \cap \cdots \cap S_{i-1}) \cup S_i) : x, w)$, $((S_1 \cap \cdots \cap S_{i-1} \cap S_i) : x, w)$, and $(S_i : x, w_i - w)$. Since $S_1 \cap \cdots \cap S_m = \emptyset$ by the rule's proviso, the final intersection is empty and a contradiction $((\square, w))$ is derived. At every step, Proposition 6.3.2 guarantees that the total weight of unsatisfied formulas is preserved; hence the rule as a whole preserves the total weight of unsatisfied soft formulas.

□

Theorem 6.3.4. Soundness and completeness. *Let T be a completed Weighted Partial Signed MaxSAT tableau for a multiset Φ of weighted signed formulas. The cost of T is m if and only if the minimum total weight of the soft formulas in Φ that are falsified by any assignment satisfying all hard formulas is m; and the cost of T is infinite if and only if no assignment satisfies all hard formulas.*

Proof. The tableau T was derived by creating a sequence of tableaux $T_0, \ldots, T_n$ ($n \geq 0$) such that T_0 is an initial tableau for Φ, $T_n = T$, and T_i was obtained by a single application of one of the expansion rules in Figure 6.3 on a branch of T_{i-1}, for $i = 1, \ldots, n$. By Proposition 6.3.1, we know that such a sequence is finite.

(Soundness:) Since T has cost m, the tableau T_n contains a saturated branch b whose total weight of formulas containing $\square$ is exactly m, while the total weight of the rest of the branches is at least m.

In every branch of T_n, the active formulas are weighted signed literals of the form $(S : x, w)$ or $(S : x, \infty)$, or formulas of the form $(\square, w)$; otherwise, further rules could be applied. For each propositional variable x, there is at most one hard literal $(S^x_{hard} : x, \infty)$ due to the application of the $\cap_{hard}$-rule. Moreover, due to the application of the $\cap^{hard}_{soft}$-rule, each active soft literal $(S : x, w)$ satisfies $S \subseteq S^x_{hard}$.

Let $S^{\cap}_x$ be the intersection of all the signs of the active soft literals in a branch b of cost m containing variable x; this intersection is not empty due to the $\square_{soft}$-rule. It also holds that $S^{\cap}_x \subseteq S^x_{hard}$. We now define an assignment v that assigns to x a value in $S^{\cap}_x$.

This assignment satisfies all active hard literals in b as well as all active soft literals. Therefore, v falsifies only those soft formulas of the form $(\square, w)$ whose total weight in branch b is m. Since every branch has at least cost m, no assignment can satisfy all hard formulas while yielding a total weight of unsatisfied soft formulas strictly less than m in a branch. Thus, the minimum total weight of unsatisfied soft formulas among the branches of T_n is exactly m.

Proposition 6.3.3 ensures that the total weight of soft formulas unsatisfied by any feasible assignment is preserved in the sequence $T_0, \ldots, T_n$. Therefore, the minimum total weight of unsatisfied soft formulas in T_0 is also m. Since T_0 consists of a single branch containing only the formulas in Φ and all the formulas are active, the minimum total weight of soft formulas that are unsatisfied in Φ by any assignment satisfying all hard formulas is m.

If the cost of T is infinite, every branch of T_n contains the symbol $\blacksquare$. By Proposition 6.3.3, no assignment can satisfy all hard formulas in Φ and there is no feasible solution.

(Completeness:) Assume that T is a completed Weighted Partial Signed MaxSAT tableau for Φ that does not have cost m. We distinguish two cases.

(i) T has a branch b of cost k, where $k < m$. Then, the total weight of the soft formulas containing the symbol $\square$ in b is k, and there exists a feasible assignment whose total weight of unsatisfied active soft formulas in b is k. By Proposition 6.3.3, this also holds for T_0, but this is in contradiction with m being the minimum total weight of the soft formulas in Φ falsified by any feasible assignment because $k < m$. Thus, any branch of T has at least cost m.

(ii) T has no branch of cost m. This is in contradiction with the fact that the minimum total weight of the unsatisfied soft formulas in Φ is m. Since the expansion rules preserve the minimum total weight of the unsatisfied soft formulas and the branches not containing the symbol $\blacksquare$ of any completed tableau only contain active soft formulas containing the symbol $\square$ or soft signed literals from which we cannot derive contradictions, T must have a saturated branch where the total weight of the soft formulas containing the symbol $\square$ is m. Thus, T has a branch of cost m.

Hence, each completed tableau T for a multiset Φ of signed formulas has cost m if the minimum total weight of the soft formulas in Φ that are falsified by any feasible assignment is m, and has infinite cost if no such assignment exists.

□

Given a completed Weighted Partial Signed MaxSAT tableau, the proof of the previous theorem also shows how to extract an optimal assignment v from a minimum-cost feasible branch b. For every variable x, the value $v(x)$ is chosen arbitrarily from $S_x^{\cap}$, where $S_x^{\cap}$ denotes the intersection of the signs of the active soft literals in b that contain occurrences of x. This assignment satisfies all hard formulas and all active soft literals, falsifying only those formulas in b of the form $(\square, w)$.

Example 6.3.5. *Figure 6.4 shows how to create a completed Weighted Partial Signed MaxSAT tableau for the multiset* $\Phi = \{\ (\{\frac{1}{2}, 1\} : \neg(x_1 \wedge x_2), 5),\ (\{\frac{1}{2}, 1\} : x_1 \vee x_2, 3),\ (\{0\} : x_1 \vee x_2, \infty)\}$*, assuming* $N = \{0, \frac{1}{2}, 1\}$*. The first tableau is the initial tableau. The second tableau applies the* $\neg$*-rule to* $(\{\frac{1}{2}, 1\} : \neg(x_1 \wedge x_2), 5)$*, the* $\vee$*-rule to*

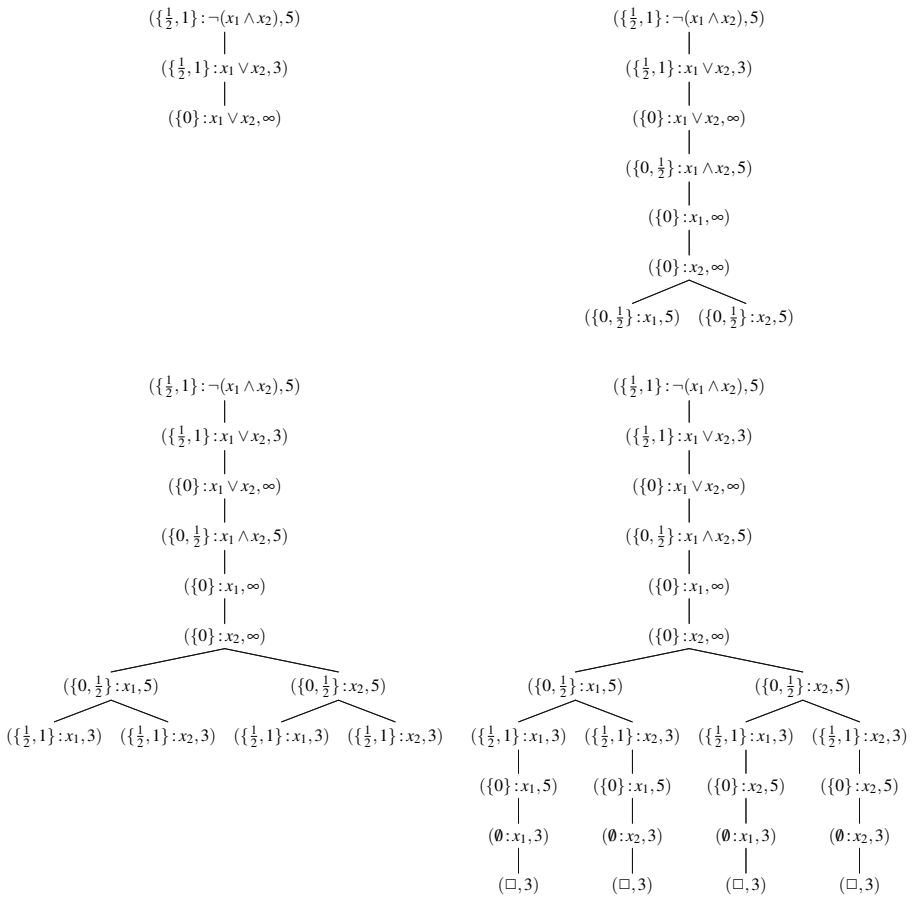


Figure 6.4: A completed Weighted Partial Signed MaxSAT tableau for $(\{\frac{1}{2},1\}:\neg(x_1\wedge x_2),5),(\{\frac{1}{2},1\}:x_1\vee x_2,3),(\{0\}:x_1\vee x_2,\infty)\}$, where $N=\{0,\frac{1}{2},1\}$.

$(\{0\}:x_1\vee x_2,\infty)$*, and the* $\wedge$*-rule to* $(\{0,\frac{1}{2}\}:x_1\wedge x_2,5)$*. The third tableau applies the* $\vee$*-rule to* $(\{\frac{1}{2},1\}:x_1\vee x_2,3)$*. The fourth tableau applies the* $\cap^{hard}_{soft}$*-rule to* $(\{0\}:x_1,\infty)$ *and* $(\{0,\frac{1}{2}\}:x_1,5)$*, to* $(\{0\}:x_1,\infty)$ *and* $(\{\frac{1}{2},1\}:x_1,3)$*, to* $(\{0\}:x_2,\infty)$ *and* $(\{0,\frac{1}{2}\}:x_2,5)$*, and to* $(\{0\}:x_2,\infty)$ *and* $(\{\frac{1}{2},1\}:x_2,3)$*. It also shows the application of the* $\emptyset$*-rule to* $(\emptyset:x_1,3)$ *and* $(\emptyset:x_2,3)$*. Since the tableau is completed and has cost 3, the minimum total weight of the soft formulas in* Φ *that are falsified by any assignment satisfying all hard formulas is also 3.*

6.4 A Signed MinSAT Tableau Calculus

This section introduces a tableau calculus for Signed MinSAT. Figure 6.5 presents its graphical representation, assuming $N = \{0, \frac{1}{2}, 1\}$ and using the symbol $\top$ to denote that a formula is satisfied. While Signed MinSAT can equivalently be formulated as the problem of maximizing the number of violated formulas, the proposed calculus is designed to compute the minimum number of formulas that are satisfied by any assignment.

As with all MaxSAT and MinSAT inference rules introduced so far, each application replaces the premises with the corresponding conclusions. Consequently, formulas in a branch are classified as either active or inactive: active formulas may be selected as premises in subsequent rule applications, whereas inactive formulas may not. The rules are designed so that the number of satisfied formulas in the premises is preserved in at least one branch and never decreases in the remaining branches.

For rules with a single premise, the introduction of a branch containing only the symbol $\top$ guarantees that the number of satisfied formulas is preserved correctly. For example, the premise $\{0\} : A \wedge B$ is satisfied whenever either A or B evaluates to 0. In the special case where both A and B evaluate to 0, only one satisfied formula should be counted, since there is only a single premise. This situation is captured by introducing a branch containing only the symbol $\top$.

Definition 6.4.1. *A Signed MinSAT tableau T for a multiset Φ of signed formulas is a tree with a finite number of branches whose nodes are labelled by a signed formula or the symbol $\top$, which denotes a formula that is satisfied. A branch is a path from the root to a leaf, and we assume that branches have a finite number of nodes.*

A tableau T for Φ is constructed by a sequence of applications of the rules in Figure 6.5. We start by constructing the initial tableau, which consists of a single branch where each node is labelled with a formula in Φ. These formulas are declared to be active. After applying an expansion rule taking as premises active formulas in a branch b, these formulas become inactive in b, and the newly added formulas are declared to be active.

Definition 6.4.2. *Let T be a Signed MinSAT tableau for a multiset Φ of signed formulas. A branch b of T is saturated when no further expansion rules can be applied to b, and T is completed when all its branches are saturated. The cost of a saturated branch is the number of occurrences of $\top$ in the branch. The cost of a completed tableau is the minimum cost among all its branches.*

Theorem 6.4.3. Soundness & completeness. *Let T be a completed Signed MinSAT tableau for a multiset Φ of signed formulas. Then, the cost of T is m if and only if the minimum number of formulas in Φ that are satisfied by any assignment is m.*

The proof of Theorem 6.4.3 follows directly from the completeness theorem for Weighted Partial MinSAT (Theorem 6.5.5) under the assumption that all formulas are soft and are assigned the same weight.

Example 6.4.4. *Figure 6.6 shows how to create a completed Signed MinSAT tableau for the multiset $\Phi = \{\{\frac{1}{2}, 1\} : \neg(x_1 \wedge x_2), \{\frac{1}{2}, 1\} : x_1 \vee x_2\}$, assuming $N = \{0, \frac{1}{2}, 1\}$. The*

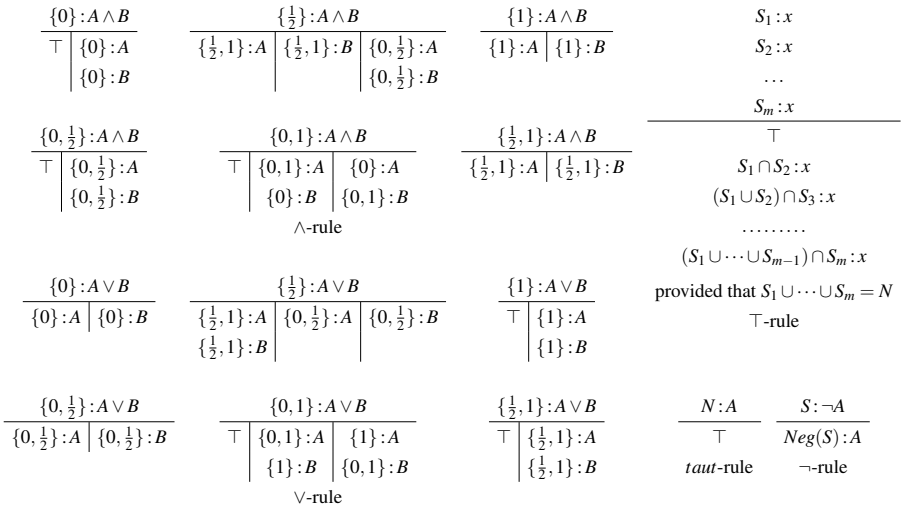


Figure 6.5: A complete tableau calculus for Signed MinSAT.

first tableau (the leftmost tableau on top) is the initial tableau, which contains one node for each formula in Φ*. The second tableau shows the application of the* $\neg$*-rule to* $\{\frac{1}{2},1\}:\neg(x_1\wedge x_2)$*, and the* $\wedge$*-rule to* $\{0,\frac{1}{2}\}:x_1\wedge x_2$*. The third tableau shows the application of the* $\vee$*-rule to* $\{\frac{1}{2},1\}:x_1\vee x_2$*. The fourth tableau shows the application of the* $\top$*-rule to* $\{0,\frac{1}{2}\}:x_1$ *and* $\{\frac{1}{2},1\}:x_1$*, and to* $\{0,\frac{1}{2}\}:x_2$ *and* $\{\frac{1}{2},1\}:x_2$*. Since the tableau is completed and the minimum cost among its branches is 1, the minimum number of formulas in* Φ *that are satisfied by any assignment is also 1. This claim follows from Theorem 6.4.3.*

6.5 A Weighted Partial Signed MinSAT Tableau Calculus

This section extends the calculus in Figure 6.5 to handle Weighted Partial Signed MinSAT instances. The resulting calculus is shown in Figure 6.7. We omit the $\wedge$-rule, as it can be derived from the $\vee$-rule and the $\neg$-rule.

For the $\vee$-, $\neg$-, and *taut*-rules, we distinguish between expansion rules with a soft premise and those with a hard premise. Rules with a single soft premise are identical to their counterparts in the unweighted setting, except that every formula is assigned the same weight w. Consequently, the weight of the premise is propagated directly to its conclusions, ensuring that whenever the premise is satisfied, exactly one formula is satisfied in one branch, whereas at least one formula is satisfied in each of the remaining branches. Conceptually, applying a soft rule with weight w is equivalent to applying the corresponding unweighted rule w times.

In contrast, rules with a hard premise differ from those in the unweighted setting. This is because their purpose is to preserve the satisfiability of the hard premise. More

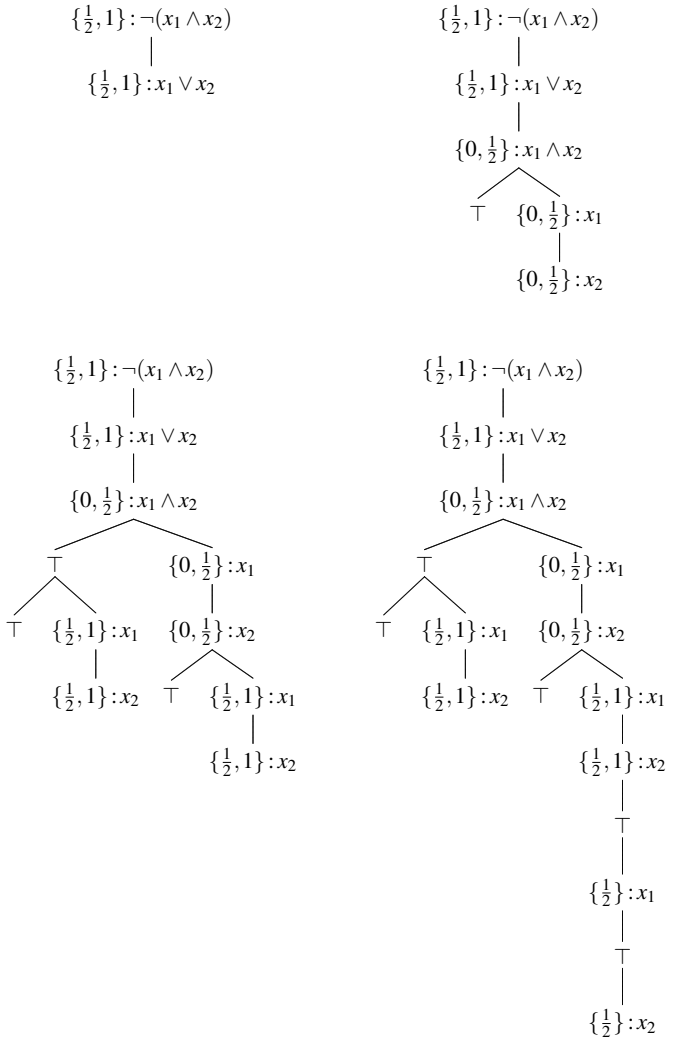


Figure 6.6: A completed Signed MinSAT tableau for the multiset of signed formulas $\{\{\frac{1}{2},1\}:\neg(x_1 \wedge x_2), \{\frac{1}{2},1\}:x_1 \vee x_2\}$, assuming $N = \{0,\frac{1}{2},1\}$.

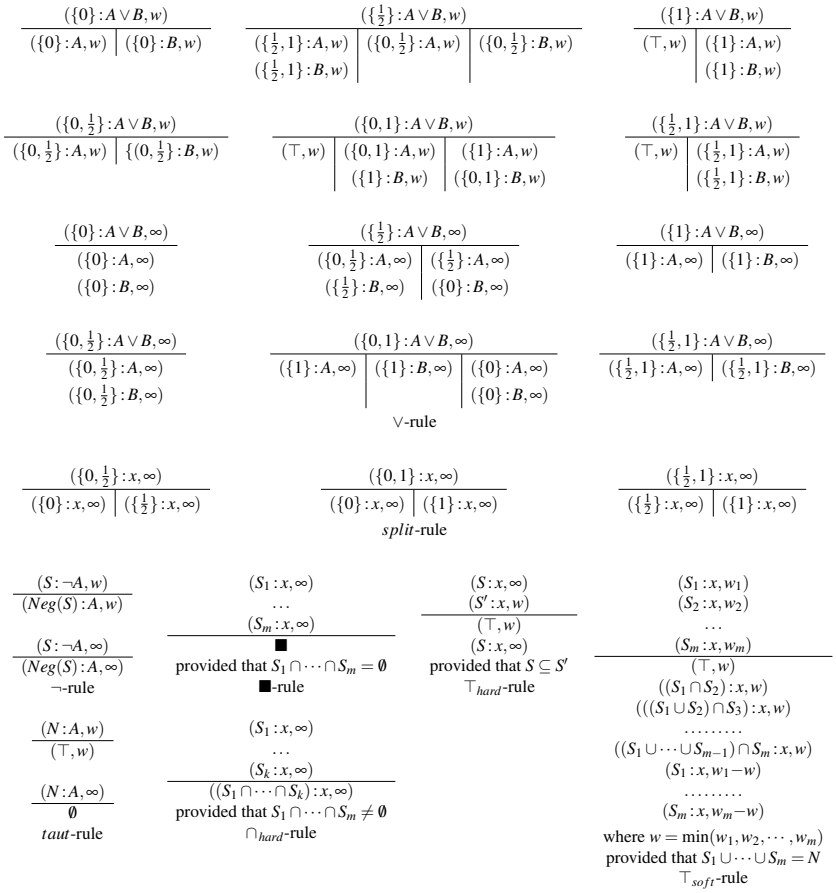


Figure 6.7: A complete tableau calculus for Weighted Partial Signed MinSAT.

precisely, the premise is satisfiable if and only if the formulas in at least one of the conclusion branches are satisfiable. In fact, these rules coincide with those of the Signed SAT calculus [49].

The $\blacksquare$-rule prunes a branch when a contradiction is detected in the hard part. Specifically, if an inconsistency arises among hard formulas, the branch is declared unsatisfiable, and no further rules are applied to it. It represents an infeasible solution.

The $\top_{hard}$-rule applies to a hard formula $(S:x,\infty)$ and a soft formula $(S':x,w)$. This rule states that if the sign of the hard formula subsumes the sign of the soft formula, then the soft formula must necessarily be satisfied with weight w. This is the reason for adding $(\top,w)$ in the conclusion of the rule. Importantly, the hard formula remains active and is therefore repeated in the conclusion.

The $\top_{soft}$-rule is an adaptation of the unweighted $\top$-rule to the weighted setting. It propagates the minimum weight among the weights of the premises and introduces compensation signed literals to ensure consistency among the weights of the premises.

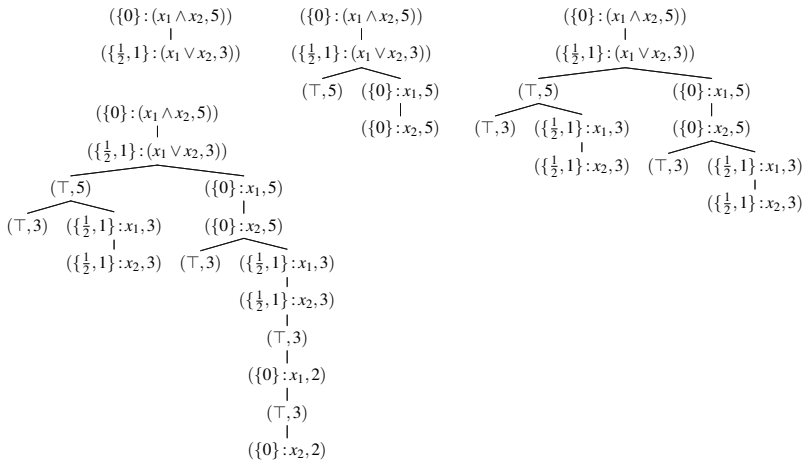


Figure 6.8: A completed Weighted Partial Signed MinSAT tableau for the multiset of signed formulas $\{\{0\}:(x_1 \wedge x_2,5),\{\frac{1}{2},1\}:(x_1 \vee x_2,3)\}$, assuming $N = \{0,\frac{1}{2},1\}$.

The minimum weight is used because the premises may involve literals with different weights. This rule aims to detect when a set of soft signed literals must necessarily be satisfied.

The $\cap_{hard}$-rule combines multiple hard literals into a single one. This simplification enables further applications of the $\top_{hard}$-rule, and is necessary to achieve completeness. Similarly, the *split*-rule is also necessary to achieve completeness. Without this rule, we cannot derive, for example, that the minimum total weight of soft formulas that are satisfied in $\{(\{\frac{1}{2},1\}:x,\infty),(\{\frac{1}{2}\}:x,1),(\{1\}:x,1)\}$ is 1. Both rules ensure that the signs of hard literals of any saturated branch are singletons.

In Weighted Partial Signed MinSAT, the cost of a saturated branch is infinite if it contains the symbol $\blacksquare$; otherwise, the total weight of the formulas of the form $(\top,w)$. The cost of a tableau is defined as the minimum cost among all its branches. An infinite cost indicates that there is no feasible solution.

Given a completed tableau T for a multiset Φ of hard and soft signed formulas, the soundness and completeness theorem (Theorem 6.5.5) establishes that the cost of T is m if and only if the minimum total weight of the soft formulas that are satisfied by any assignment satisfying all hard formulas is m.

Example 6.5.1. *Figure 6.8 shows how to create a completed Weighted Partial Signed MinSAT tableau for the multiset* $\Phi = \{(\{0\}:x_1 \wedge x_2,5),(\{\frac{1}{2},1\}:x_1 \vee x_2,3)\}$*, assuming* $N = \{0,\frac{1}{2},1\}$*. The first tableau (the leftmost tableau on top) is the initial tableau, which contains one node for each formula in* Φ*. The second tableau shows the application of the* $\wedge$*-rule to* $(\{0\}:x_1 \wedge x_2,5)$*. The third tableau shows the application of the* $\vee$*-rule to* $(\{\frac{1}{2},1\}:x_1 \vee x_2,3)$*. The fourth tableau shows the application of the* $\top_{soft}$*-rule to* $(\{0\}:x_1,5)$ *and* $(\{\frac{1}{2},1\}:x_1,3)$*, and to* $(\{0\}:x_2,5)$ *and* $(\{\frac{1}{2},1\}:x_2,3)$*. Since the tableau is completed and the minimum cost among its branches is 3, the minimum total*

weight of the soft formulas in Φ that are satisfied by any assignment satisfying all hard formulas is also 3.

Before proving the soundness and completeness of the calculus for Weighted Partial Signed MinSAT, we need to prove the following propositions.

Proposition 6.5.2. *A Weighted Partial Signed MinSAT tableau for a multiset Φ of weighted signed formulas is completed in a finite number of steps.*

Proof. We first create an initial tableau and then apply expansion rules in the newly created branches until they become saturated. The $\vee$- and $\neg$-rule reduce the number of connectives in all conclusions w.r.t. the premise, and the other rules do not increase the number of connectives in the conclusions. The application of the *taut*-rule, $\blacksquare$-rule, $\cap_{hard}$-rule, $\top_{hard}$-rule and $\top_{soft}$-rule reduces the number of signed literals in the conclusions w.r.t. the number of signed literals in the premises if we consider that a formula of the form $(S:x,w)$ is equivalent to w copies of $(S:x,1)$. The *split*-rule reduces the size of the signs and does not increase the number of signed literals. Since we began with a finite number of connectives and signed literals, and no rule increases the size of the signs, these rules can only be applied a finite number of times. □

Proposition 6.5.3. *Given the multisets of weighted signed literals $\{(S_1 : x,w_1),(S_2 : x,w_2)\}$ and $\{((S_1 \cap S_2) : x,w),((S_1 \cup S_2) : x,w),(S_1 : x,w_1-w),(S_2 : x,w_2-w)\}$, where $w = \min(w_1,w_2)$, it holds that the total weight of the signed literals satisfied by an assignment v is the same in both multisets.*

Proof. We distinguish four cases:

- v satisfies $S_1 : x$ and $S_2 : x$: Since $v(x) \in S_1$ and $v(x) \in S_2$, v satisfies both $(S_1 \cup S_2) : x$ and $(S_1 \cap S_2) : x$. Thus, v satisfies all formulas in both multisets. Since $w_1 = w + (w_1 - w)$ and $w_2 = w + (w_2 - w)$, the total weight of satisfied formulas is preserved between both multisets.

- v satisfies $S_1 : x$ and does not satisfy $S_2 : x$: Since $v(x) \in S_1$ and $v(x) \notin S_2$, v does not satisfy $(S_1 \cap S_2) : x$ but satisfies $(S_1 \cup S_2) : x$. Thus, v only satisfies formula $(S_1 : x,w_1)$ in the first multiset, and formulas $((S_1 \cup S_2) : x,w),(S_1 : x,w_1-w)$ in the second multiset. Since $w_1 = w + (w_1 - w)$, the total weight of satisfied formulas is preserved between both multisets.

- v does not satisfy $S_1 : x$ and satisfies $S_2 : x$: Since $v(x) \notin S_1$ and $v(x) \in S_2$, v does not satisfy $(S_1 \cap S_2) : x$ but satisfies $(S_1 \cup S_2) : x$. Thus, v only satisfies formula $(S_2 : x,w_2)$ in the first multiset, and formulas $((S_1 \cup S_2) : x,w),(S_2 : x,w_2-w)$ in the second multiset. Since $w_2 = w + (w_2 - w)$, the total weight of satisfied formulas is preserved between both multisets.

- v does not satisfy $S_1 : x$ and $S_2 : x$: Since $v(x) \notin S_1$ and $v(x) \notin S_2$, v does not satisfy neither $(S_1 \cap S_2) : x$ nor $(S_1 \cup S_2) : x$. Thus, v falsifies all formulas in both multisets. Since the total weight of satisfied formulas is 0, this weight is preserved between both multisets.

□

Proposition 6.5.4. *The $\vee$-rule, $\neg$-rule, taut-rule, $\blacksquare$-rule, $\cap_{hard}$-rule, $\top_{hard}$-rule, $\top_{soft}$-rule and split-rule are sound in the sense that the total weight of satisfied formulas in the premises is preserved in at least one branch and does not decrease in the remaining branches when the premises are soft. In addition, these rules preserve the satisfiability between the premises and the conclusions in at least one branch when the premises are hard.*

Proof. To prove the soundness of the $\vee$-rule, when the premise is soft, we must demonstrate that if an assignment v satisfies the single premise with weight w, there is one branch in the conclusions where v satisfies one formula with weight w, and the total weight of formulas satisfied by v in the rest of branches (if any) is at least w. When the premise is hard, we must demonstrate that the satisfiability is preserved between the premise and the conclusions. Due to the fact that all the cases are proved similarly, we only prove that it holds for the rules with premise $(\{0\}:A\vee B,w)$ and $(\{0\}:A\vee B,\infty)$. In the first case, an assignment v satisfies $(\{0\}:A\vee B,w)$ if and only if $v(A)=v(B)=0$. In this case, both branches are satisfied. Moreover, $(\{0\}:A\vee B,w)$ is unsatisfied when $v(A)\neq 0$ or $v(B)\neq 0$. In this case, at least one branch is unsatisfied. Thus, the total weight of satisfied formulas is preserved between the premise and the conclusions of the rule.

When the premise is hard, an assignment v satisfies $(\{0\}:A\vee B,\infty)$ if and only if $v(A)=v(B)=0$. Thus, the satisfiability is preserved between the premise and the conclusions of the rule, because the branch becomes unsatisfied only when $v(A)\neq 0$ or $v(B)\neq 0$.

The soundness of the *taut*-rule, for both hard and soft formulas, follows from the fact that $N:A$ is a tautology, and the soundness of the $\neg$-rule follows from the interpretation of the negation connective.

The soundness of the $\blacksquare$-rule follows from the fact that the hard formulas cannot be simultaneously satisfied when the intersection of the signs of a set of literals containing variable x is empty. In this case, any assignment unsatisfies at least one literal.

The soundness of the $\cap_{hard}$-rule follows from the fact that a set of literals containing variable x is satisfied by an assignment v if and only if the literal containing variable x whose sign is the intersection of all their signs is satisfied by v.

The soundness of the $\top_{hard}$-rule follows from the fact that if hard formula $(S:x,\infty)$ is satisfied, $(S':x,w)$ must be necessarily satisfied because $S\subseteq S'$. Therefore, $(S':x,w)$ can be replaced with $(\top,w)$.

To prove the soundness of the $\top_{soft}$-rule, when there are m premises, we obtain the conclusion by applying Proposition 6.5.3 a total of $m-1$ times. First, we apply Proposition 6.5.3 to $(S_1:x,w_1),(S_2:x,w_2)$ and replace them with $((S_1\cup S_2):x,w)$, $((S_1\cap S_2):x,w)$, $(S_1:x,w_1-w)$ and $(S_2:x,w_2-w)$, where $w_1\geq w$ and $w_2\geq w$. In the i-th application, we apply proposition 6.5.3 to $((S_1\cup S_2\cdots\cup S_{i-1}):x,w),(S_i:x,w_i)$ and replace them with $(S_1\cup\cdots\cup S_i):x$, $((S_1\cup\cdots\cup S_{i-1})\cap S_i):x$ and $(S_i:x,w_i-w)$, where $w_i\geq w$. Since $S_1\cup\cdots\cup S_m=N$, we derive $(\top,w)$ in the last application because $(N:x,w)$ is a tautology.

The soundness of the *split*-rule follows from the fact that an assignment satisfies $\{i_1,i_2\}:x$ if and only if it satisfies $\{i_1\}:x$ or $\{i_2\}:x$.

□

Theorem 6.5.5. Soundness and completeness. *Let T be a completed Signed MinSAT tableau for a multiset Φ of signed formulas. Then, the cost of T is m if and only if the minimum total weight of the soft formulas in Φ that are satisfied by any assignment satisfying all hard formulas is m; and the cost of T is infinite if and only if no assignment satisfies all hard formulas.*

Proof. (*Soundness:*) T was derived by creating a sequence of tableaux $T_0,\ldots,T_n$ ($n \geq 0$) such that T_0 is an initial tableau for Φ, $T_n = T$, and T_i was obtained by a single application of one of the expansion rules in Figure 6.7 on a branch of T_{i-1}, for $i = 1,\ldots,n$. By proposition 6.5.2, we know that such a sequence is finite. Since T has cost m, T_n contains one saturated branch b where the total weight of formulas of the form $(\top, w)$ is m, and that weight is at least m in the rest of the branches. Moreover, the active formulas in every branch of T_n are hard signed literals, soft signed literals, or formulas of the form $(\top, w)$. For each propositional variable x, the union of the signs occurring in soft literals, denoted by $S_x^{\cup}$, is different from N; otherwise, we could yet apply the $\top_{soft}$-rule and T_n would not be completed. Moreover, any active hard literal is of the form $(\{i\} : x, \infty)$ due to the application of the *split*-rule.

For every variable x in a branch of T_n containing an active hard literal $(\{i\} : x, \infty)$, any assignment v such that $v(x) = i$ satisfies $(\{i\} : x, \infty)$, all formulas of the form $(\top, w)$, and does not satisfy any active soft literal containing variable x due to the application of the $\top_{hard}$-rule. If there is no active hard literal containing variable x, any assignment v such that $v(x) \in N \setminus S_x^{\cup}$ does not satisfy any active soft literal containing variable x. Since $(\top, w)$ is always satisfied and the total weight of formulas of the form $(\top, w)$ is at least m, the total weight of active soft formulas satisfied by v cannot be less than m in a branch of T_n. Thus, a complete assignment v' constructed in this way satisfies the hard literals and the formulas of the form $(\top, w)$, and unsatisfies all soft literals. Thus, the minimum total weight of active soft formulas satisfied among the branches of T_n is m, which is the minimum cost of T_n.

If v' is derived from a branch of minimum cost, it is an optimal assignment: proposition 6.5.4 ensures that the minimum total weight of the satisfied soft active formulas, while preserving the satisfiability of hard formulas, is preserved in the sequence of tableaux $T_0,\ldots,T_n$. Therefore, the minimum total weight of the satisfied soft formulas in the single branch of T_0 is also m. As T_0 comprises a single branch containing only the formulas in Φ and all these formulas are active, we conclude that the total weight of the satisfied soft formulas in Φ is m.

(*Completeness*): Assume that there is a completed Signed MinSAT tableau T for Φ that does not have cost m. We distinguish two cases:

(i) T has a branch b of cost k, where $k < m$. Then, the total weight of the soft formulas containing the symbol $\top$ in b is k, and there exists a feasible assignment whose total weight of satisfied active soft formulas in b is k. By Proposition 6.5.4, this also holds for T_0, but this is in contradiction with m being the minimum total weight of the soft formulas in Φ satisfied by any feasible assignment because $k < m$. Thus, any branch of T has at least cost m.

(ii) T has no branch of cost m. This is in contradiction with the fact that the minimum total weight of the satisfied soft formulas in Φ is m. Since the expansion rules preserve the minimum total weight of the satisfied soft formulas and the branches not

containing the symbol ■ of any completed tableau only contain active soft formulas containing the symbol $\top$ or soft signed literals that can be unsatisfied, T must have a saturated branch where the total weight of the soft formulas containing the symbol $\top$ is m. Thus, T has a branch of cost m.

Hence, each completed tableau T for a multiset Φ of signed formulas has cost m if the minimum total weight of soft formulas in Φ that are satisfied by any feasible assignment is m. □

Given a completed Signed MinSAT tableau, an optimal assignment can be extracted by selecting a branch b of minimum cost. The idea, as can be seen in the proof, is to define an assignment that satisfies the hard active formulas and does not satisfy any active soft formulas in b except for the formulas containing $\top$ that are always satisfied.

6.6 Conclusions

We have defined tableau calculi for Weighted Partial Signed MaxSAT and Weighted Partial Signed MinSAT, and proved their soundness and completeness. Since the expansion rules depend on the cardinality of the truth value set and lack uniformity, the calculi operate over the fixed three-valued domain $N = \{0, \frac{1}{2}, 1\}$. These calculi extend the proof-theoretic support for optimization problems from regular logic to the more general setting of signed logic, where signs are arbitrary subsets of the truth value set.

Chapter 7

Conclusions and Future Work

This chapter summarizes the main contributions achieved throughout this thesis, contextualizing them within the scope of the original research objectives. Furthermore, it outlines several promising research directions and open problems, highlighting potential avenues for future work.

7.1 Conclusions

The primary objective of this thesis was to strengthen the proof-theoretic foundations of Boolean and many-valued optimization while developing practical exact solving techniques grounded in these theoretical advances. The results presented throughout this dissertation achieve this objective by introducing new calculi, exact algorithms, and encodings spanning both Boolean and many-valued formalisms.

A highly optimized exact variable elimination solver for Weighted Partial MaxSAT, named MSElim, has been designed, implemented, and empirically evaluated. Experimental results on benchmark instances from the MaxSAT Evaluation demonstrate that exploiting the MaxSAT resolution rule within a carefully engineered solver yields an effective MaxSAT solving approach. In particular, the results on Bounded Variable Elimination (BVE) preprocessing suggest that the MaxSAT resolution-based BVE implemented in MSElim is a promising preprocessing technique for MaxSAT solvers.

Beyond the Boolean setting, the proof-theoretic foundations of many-valued optimization have been significantly extended. The first sound and complete resolution calculus for Weighted Regular MaxSAT has been established, providing a formal proof system together with the first exact variable elimination algorithm for this logic. Furthermore, sound and complete tableau calculi have been developed for both Weighted Partial Regular MaxSAT and Weighted Partial Regular MinSAT, considerably extending the basis for optimization in regular logic.

These tableau calculi served as the starting point for a novel Mixed-Integer Linear Programming (MILP) encoding for Regular MaxSAT and Regular MinSAT. This encoding enables state-of-the-art MILP solvers to efficiently compute optimal solutions, offering a practical solving framework that complements the underlying theoretical

contributions.

Finally, these theoretical advances have been lifted to signed logic, yielding the first sound and complete tableau calculi for Weighted Partial Signed MaxSAT and Weighted Partial Signed MinSAT.

Overall, this thesis demonstrates that proof-theoretic techniques originally developed for Boolean optimization can be successfully extended to richer many-valued logics. At the same time, it shows that these techniques also enable improved exact solving methods for Boolean MaxSAT through an exact variable elimination algorithm based on MaxSAT resolution.

By combining new calculi, exact algorithms, and MILP encodings, this work strengthens the proof-theoretic foundations and the collection of exact solving techniques for Boolean and many-valued optimization, bringing the latter much closer to the level of maturity achieved in the Boolean setting.

7.2 Future Work

The contributions presented in this thesis advance the theoretical and algorithmic foundations of Boolean and many-valued optimization while also opening new opportunities for further research. Building upon these results, the following research directions are proposed:

- The developed Boolean variable elimination solver, MSElim, could be exploited to design new preprocessing and inprocessing techniques, potentially improving the scalability and robustness of state-of-the-art MaxSAT solvers.
- Identifying suitable strategies for selecting the clauses that remain active after saturating a variable may pave the way for the development of an exact variable elimination algorithm for Regular MinSAT based on the proposed Regular MaxSAT resolution rule.
- Extending the tableau calculi developed for Regular MaxSAT and Regular MinSAT to first-order logic constitutes a natural theoretical continuation of this work.
- The experimental evaluation of the MILP encoding was conducted on unweighted random Max-2SAT and Max-3SAT instances. Extending it to weighted partial instances and structured benchmarks would provide a more complete picture of its practical performance.
- From an empirical perspective, an important next step is to compare the proposed MILP approach with state-of-the-art MaxSAT and MinSAT solvers on a broad range of combinatorial optimization problems, offering a clearer assessment of its strengths and limitations.
- A longer-term research direction concerns the integration of the proof-theoretic techniques developed in this thesis with learning-based methods for exact optimization. Rather than replacing symbolic reasoning, machine learning could

be employed to guide it by predicting effective variable elimination orders, selecting inference rules, estimating the usefulness of derived clauses, or identifying promising branching strategies. Since the underlying calculi and algorithms preserve the exactness independently of these heuristic decisions, such learned guidance may improve solver performance without compromising correctness. Exploring the interaction between symbolic reasoning and machine learning may ultimately lead to more scalable exact MaxSAT and MinSAT solvers capable of handling larger and more structurally diverse optimization instances.

Bibliography

[1] A. Abramé and D. Habet. Ahmaxsat: Description and Evaluation of a branch and bound Max-SAT solver. *J. Satisf. Boolean Model. Comput.*, 9(1):89–128, 2014.

[2] C. Ansótegui, M. L. Bonet, J. Levy, and F. Manyà. The logic behind weighted CSP. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI) , Hyderabad, India, January 6-12, 2007*, pages 32–37.

[3] C. Ansótegui, M. L. Bonet, J. Levy, and F. Manyà. A complete resolution calculus for signed Max-SAT. In *Proceedings of the 37th International Symposium on Multiple-Valued Logic (ISMVL), Oslo, Norway*, pages 22–27, 2007.

[4] C. Ansótegui, M. L. Bonet, J. Levy, and F. Manyà. Inference rules for high-order consistency in weighted CSP. In *Proceedings of the 22nd Conference on Artificial Intelligence (AAAI), Vancouver, British Columbia, Canada*, pages 167–172, 2007.

[5] C. Ansótegui, M. L. Bonet, J. Levy, and F. Manyà. Resolution procedures for multiple-valued optimization. *Information Sciences*, 227:43–59, 2013.

[6] C. Ansótegui, F. Manyà, J. Ojeda, J. M. Salvia, and E. Torres. Incomplete MaxSAT approaches for combinatorial testing. *Journal of Heuristics*, 28(4):377–431, 2022.

[7] J. Argelich, C. M. Li, F. Manyà, and J. R. Soler. Clause tableaux for Maximum and Minimum Satisfiability. *Logic Journal of the IGPL*, 29(1):7–27, 2021.

[8] J. Argelich, C. M. Li, F. Manyà, and Z. Zhu. Many-valued MinSAT solving. In *Proceedings, 44th International Symposium on Multiple-Valued Logics (ISMVL), Bremen, Germany*, pages 32–37, 2014.

[9] F. Bacchus. MaxHS in the 2020 MaxSAT Evaluation. In *MaxSAT Evaluation 2020: Solver and Benchmark Descriptions*, pages 19–20, 2020.

[10] F. Bacchus, M. Järvisalo, and R. Martins. Maximum Satisfiability. In *Handbook of Satisfiability*, pages 929–991. IOS Press, 2021.

[11] B. Beckert, R. Hähnle, and F. Manya. The SAT problem of signed CNF formulas. In *Labelled Deduction*, pages 59–80. Springer, 2000.

[12] R. Béjar and F. Manyà. Solving the round robin problem using propositional logic. In *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI), Austin/TX, USA*, pages 262–266, 2000.

[13] J. Berg, B. Bogaerts, J. Nordström, A. Oertel, and D. Vandesande. Certified core-guided maxsat solving. In *Proceedings 29th International Conference on Automated Deduction (CADE), Rome, Italy*, pages 1–22, 2023.

[14] J. Berg, M. Järvisalo, R. Martins, A. Niskanen, and T. Paxian. MaxSAT Evaluation 2024: Solver and benchmark descriptions. 2024.

[15] J. Berg, M. Järvisalo, R. Martins, A. Niskanen, and T. Paxian. MaxSAT Evaluation 2024 website. `https://maxsat-evaluations.github.io/2024/`, 2024.

[16] A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu. Bounded model checking. *Advances in Computers*, 58:117–148, 2003.

[17] A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2021.

[18] A. Biere, M. Järvisalo, and B. Kiesl. Preprocessing in SAT solving. In *Handbook of Satisfiability - Second Edition*, pages 391–435. IOS Press, 2021.

[19] A. Biere and W. Kunz. SAT and ATPG: boolean engines for formal hardware verification. In *Proceedings of International Conference on Computer-aided Design (ICCAD), San Jose, California, USA, November 10-14*, pages 782–785, 2002.

[20] M. Bofill, J. Coll, M. Garcia, J. Giráldez-Cru, G. Pesant, J. Suy, and M. Villaret. Constraint solving approaches to the business-to-business meeting scheduling problem. *Journal of Artificial Intelligence Research*, 74:263–301, 2022.

[21] I. Bonacina, M. L. Bonet, and J. Levy. Weighted, circular and semi-algebraic proofs. *Journal of Artificial Intelligence Research*, 79:447–482, 2024.

[22] M. L. Bonet, S. Buss, A. Ignatiev, A. Morgado, and J. Marques-Silva. Propositional proof systems based on Maximum Satisfiability. *Artificial Intelligence*, 300:103552, 2021.

[23] M. L. Bonet and J. Levy. Equivalence between systems stronger than resolution. In *Proceedings of the 23rd International Conference on Theory and Applications of Satisfiability Testing, (SAT), Alghero, Italy*, pages 166–181, 2020.

[24] M. L. Bonet, J. Levy, and F. Manyà. A complete calculus for Max-SAT. In *Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing (SAT), Seattle, USA*, pages 240–251, 2006.

[25] M. L. Bonet, J. Levy, and F. Manyà. Resolution for Max-SAT. *Artificial Intelligence*, 171(8–9):240–251, 2007.

[26] M. Češka, J. Matyáš, V. Mrazek, and T. Vojnar. Satisfiability solving meets evolutionary optimisation in designing approximate circuits. In *Proceedings of the 23rd International Conference on Theory and Applications of Satisfiability Testing (SAT), Alghero, Italy*, pages 481–491, 2020.

[27] M. S. Cherif, D. Habet, and A. Abramé. Understanding the power of Max-SAT resolution through up-resilience. *Artificial Intelligence*, 289:103397, 2020.

[28] E. Clarke, D. Kroening, and F. Lerda. A tool for checking ANSI-C programs. In *Proceedings of the 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), Barcelona, Spain*, pages 168–176, 2004.

[29] E. M. Clarke, A. Biere, R. Raimi, and Y. Zhu. Bounded model checking using Satisfiability solving. *Formal Methods in System Design*, 19(1):7–34, 2001.

[30] J. Coll, C. M. Li, S. Li, D. Habet, and F. Manyà. Solving weighted Maximum Satisfiability with branch and bound and clause learning. *Computers and Operations Research*, 183:107195, 2025.

[31] J. Coll, C. M. Li, F. Manyà, and E. Yangin. A complete tableau calculus for the Regular MaxSAT problem. In *Proceedings of the 25th International Conference of the Catalan Association for Artificial Intelligence (CCIA), Món Sant Benet, Spain*, pages 359–368, 2023.

[32] J. Coll, C. M. Li, F. Manyà, and E. Yangin. MaxSAT resolution for Regular propositional logic. *International Journal of Approximate Reasoning*, 162:109010, 2023.

[33] J. Coll, C. M. Li, F. Manyà, and E. Yangin. A complete tableau calculus for non-clausal Regular MinSAT. In *Proceedings of the 26th International Conference of the Catalan Association for Artificial Intelligence (CCIA), Barcelona, Spain*, pages 278–287, 2024.

[34] J. Coll, C. M. Li, F. Manyà, and E. Yangin. A tableau calculus for non-clausal Regular MaxSAT. In *Proceedings of the 54th International Symposium on Multiple-Valued Logic (ISMVL), Brno, Czech Republic*, pages 137–142, 2024.

[35] J. Coll, C. M. Li, F. Manyà, and E. Yangin. Complete tableau calculi for Regular MaxSAT and Regular MinSAT. *Cognitive Systems Research*, 90:101319, 2025.

[36] J. Coll, C. M. Li, F. Manyà, and E. Yangin. A complete tableau calculus for Signed MaxSAT. In *Proceedings of the 55th International Symposium on Multiple-Valued Logic (ISMVL), Montreal, QC, Canada*, pages 38–43, 2025.

[37] M. D'Agostino. Tableaux methods for classical propositional logic. In *Handbook of Tableau Methods*, pages 45–123. Kluwer, 1999.

[38] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM (JACM)*, 7(3):201–215, 1960.

[39] N. Eén and A. Biere. Effective preprocessing in SAT through variable and clause elimination. In *Proceedings Theory and Applications of Satisfiability Testing, 8th International Conference (SAT) 2005, St. Andrews, UK, June 19-23*, pages 61–75.

[40] N. Eén and N. Sörensson. An extensible SAT-solver. In *Proceedings Theory and Applications of Satisfiability Testing, 6th International Conference (SAT), Santa Margherita Ligure, Italy, May 5-8*, pages 502–518, 2003.

[41] R. Ehlers, K. Treutler, and V. Wesling. SAT solving with fragmented Hamiltonian path constraints for wire arc additive manufacturing. In *Proceedings of the 23rd International Conference on Theory and Applications of Satisfiability Testing (SAT), Alghero, Italy*, pages 492–500, 2020.

[42] T. Eibach, E. Pilz, and G. Völkel. Attacking Bivium using SAT solvers. In *Proceedings of the 11th International Conference on Theory and Applications of Satisfiability Testing (SAT), Guangzhou, China*, pages 63–76, 2008.

[43] M. Finger and S. Preto. Polyhedral semantics and the tractable approximation of Łukasiewicz infinitely-valued logic. *Journal of Logic and Computation*, 35(5), 2025.

[44] G. Fiorino. New tableau characterizations for non-clausal MaxSAT problem. *Logic Journal of the IGPL*, 30(3):422–436, 2022.

[45] G. Fiorino. A non-clausal tableau calculus for MinSAT. *Information Processing Letters*, 173:106167, 2022.

[46] G. Fiorino. New tableau characterizations for non-clausal regular MinSAT and MaxSAT problems. *Logic Journal of the IGPL*, 33(6), 2025.

[47] C. P. Gomes, B. Selman, K. McAloon, and C. Tretkoff. Randomization in backtrack search: Exploiting heavy-tailed profiles for solving hard scheduling problems. In *Proceedings of the 4th International Conference on Artificial Intelligence Planning Systems (AIPS), Pittsburg, PA, USA*, pages 208–213, 1998.

[48] L. Guerra e Silva, L. M. Silveira, and J. Marques-Silva. Algorithms for solving Boolean Satisfiability in combinational circuits. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE), Munich, Germany*, pages 526–530, 1999.

[49] R. Hähnle. *Automated Deduction in Multiple-Valued Logics*, volume 10 of *International Series of Monographs in Computer Science*. Oxford University Press, 1994.

[50] R. Hähnle. Many-valued logic and mixed integer programming. *Annals of Mathematics and Artificial Intelligence*, 12:231–263, 1994.

[51] Z. Haníková, F. Manyà, and A. Vidal. The MaxSAT problem in the real-valued MV-algebra. In *Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX)*, pages 386–404, 2023.

[52] M. J. H. Heule. Schur number five. In *Proceedings of the 32nd Conference on Artificial Intelligence (AAAI), New Orleans, Louisiana, USA*, pages 6598–6606, 2018.

[53] M. J. H. Heule, O. Kullmann, and V. W. Marek. Solving very hard problems: Cube-and-Conquer, a hybrid SAT solving method. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI), Melbourne, Australia*, pages 4864–4868, 2017.

[54] Q. Huangfu and J. A. J. Hall. Parallelizing the dual revised simplex method. *Mathematical Programming Computation*, 10(1):119–142, 2018.

[55] A. Ignatiev, A. Morgado, and J. Marques-Silva. On reducing maximum independent set to Minimum Satisfiability. In *Proceedings 17th International Conference on Theory and Applications of Satisfiability Testing (SAT), Vienna, Austria*, pages 103–120, 2014.

[56] D. Kaiss, M. Skaba, Z. Hanna, and Z. Khasidashvili. Industrial strength SAT-based alignability algorithm for hardware equivalence verification. In *Proceedings of the 7th International Conference on Formal Methods in Computer-Aided Design (FMCAD), Austin, Texas, USA*, pages 20–26, 2007.

[57] H. Kautz. Deconstructing planning as Satisfiability. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI), Boston, Massachusetts*, pages 1524–1526, 2006.

[58] H. A. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI), Portland, OR, USA*, pages 1194–1201, 1996.

[59] B. Konev and A. Lisitsa. Computer-aided proof of Erdős discrepancy properties. *Artificial Intelligence*, 224:103–118, 2015.

[60] J. Larrosa and F. Heras. Resolution in Max-SAT and its relation to local consistency in weighted CSPs. In L. P. Kaelbling and A. Saffiotti, editors, *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI), Edinburgh, Scotland, UK*, pages 193–198, 2005.

[61] J. Larrosa, F. Heras, and S. de Givry. A logical approach to efficient Max-SAT solving. *Artificial Intelligence*, 172(2-3):204–233, 2008.

[62] C. Li, Z. Xu, J. Coll, F. Manyà, D. Habet, and K. He. Combining clause learning and branch and bound for MaxSAT. In *Proceedings of the 27th International Conference on Principles and Practice of Constraint Programming (CP), Montpellier, France*, pages 38:1–38:18, 2021.

[63] C. Li, Z. Xu, J. Coll, F. Manyà, D. Habet, and K. He. Boosting branch-and-bound MaxSAT solvers with clause learning. *AI Communications*, 35(2):131–151, 2022.

[64] C. M. Li and F. Manya. MaxSAT, hard and soft constraints. In *Handbook of Satisfiability*, pages 613–631. IOS Press, 2009.

[65] C. M. Li and F. Manyà. An exact inference scheme for MinSAT. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI), Buenos Aires, Argentina*, pages 1959–1965, 2015.

[66] C. M. Li and F. Manyà. Inference in MaxSAT and MinSAT. In *Proceedings The Logic of Software. A Tasting Menu of Formal Methods: Essays Dedicated to Reiner Hähnle on the Occasion of His 60th Birthday*, pages 350–369, 2022.

[67] C. M. Li, F. Manyà, N. O. Mohamedou, and J. Planes. Exploiting cycle structures in Max-SAT. In *Proceedings of the 12th International Conference Theory and Applications of Satisfiability Testing - (SAT) 2009, Swansea, UK.*, pages 467–480, 2009.

[68] C. M. Li, F. Manyà, and J. Planes. New inference rules for Max-SAT. *Journal of Artificial Intelligence Research*, 30:321–359, 2007.

[69] C. M. Li, F. Manyà, and J. R. Soler. A clause tableaux calculus for MaxSAT. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI), New York, USA*, pages 766–772, 2016.

[70] C. M. Li, F. Manyà, and J. R. Soler. Clausal form transformation in MaxSAT. In *Proceedings of the 49th International Symposium on Multiple-Valued Logic (ISMVL), Fredericton, Canada*, pages 132–137, 2019.

[71] C. M. Li, F. Manyà, and J. R. Soler. A tableau calculus for non-clausal Maximum Satisfiability. In *Proceedings of the 28th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX), London, UK*, pages 58–73, 2019.

[72] C. M. Li, F. Manyà, J. R. Soler, and A. Vidal. Clausal forms in MaxSAT and MinSAT. *International Journal of Computational Intelligence Systems*, 15(1):97, 2022.

[73] C. M. Li, Z. Zhu, F. Manyà, and L. Simon. Minimum Satisfiability and its applications. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI), Barcelona, Spain*, pages 605–610, 2011.

[74] C. M. Li, Z. Zhu, F. Manyà, and L. Simon. Optimizing with Minimum Satisfiability. *Artificial Intelligence*, 190:32–44, 2012.

[75] S. Li, C. Li, M. Luo, J. Coll, D. Habet, and F. Manyà. A new variable ordering for in-processing bounded variable elimination in SAT solvers. In *Proceedings of the 32nd International Joint Conference on Artificial Intelligence (IJCAI), Macao, SAR, China*, pages 1979–1987, 2023.

[76] I. Lynce and J. a. Marques-Silva. Efficient haplotype inference with Boolean Satisfiability. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI), Boston, Massachusetts*, page 104–109, 2006.

[77] I. Lynce and J. a. Marques-Silva. SAT in bioinformatics: Making the case with haplotype inference. In *Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing, (SAT), Seattle, WA, USA*, page 136–141, 2006.

[78] F. Manyà, S. Negrete, C. Roig, and J. R. Soler. Solving the team composition problem in a classroom. *Fundamamenta Informaticae*, 174(1):83–101, 2020.

[79] F. Maric and P. Janicic. Formalization of abstract state transition systems for SAT. *Logical Methods in Computer Science*, 7, 2011.

[80] J. Marques-Silva, J. Argelich, A. Graça, and I. Lynce. Boolean lexicographic optimization: algorithms & applications. *Annals of Matematics and Artificial Intelligence*, 62(3-4):317–343, 2011.

[81] J. Marques-Silva, I. Lynce, and S. Malik. Conflict-driven clause learning SAT solvers. *In Handbook of Satisfiability*, pages 131–153, 2009.

[82] R. Martins, V. M. Manquinho, and I. Lynce. Open-WBO: A modular MaxSAT solver. In *Proceedings of SAT 2014*, pages 438–445, 2014.

[83] A. Mihal and S. Teig. A constraint satisfaction approach for programmable logic detailed placement. In *Proceedings of the 16th International Conference on Theory and Applications of Satisfiability Testing (SAT), Helsinki, Finland*, page 208–223, 2013.

[84] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference (DAC), Las Vegas, NV, USA, June 18-22*, pages 530–535, 2001.

[85] T. Paxian and B. Becker. Pacose: An iterative SAT-based MaxSAT solver. In *MaxSAT Evaluation 2020: Solver and Benchmark Descriptions*, page 12, 2020.

[86] J. Rintanen. Engineering efficient planners with SAT. In *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI), Montpellier, France*, page 684–689, 2012.

[87] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965.

[88] E. Rollon and J. Larrosa. Proof complexity for the Maximum Satisfiability problem and its use in SAT refutations. *Journal of Logic and Computation*, 32(7):1401–1435, 2022.

[89] R. Smullyan. *First-Order Logic*. Dover Publications, New York, second corrected edition, 1995.

[90] N. Sörensson and N. Eén. Minisat 2.1 and Minisat++ 1.0-sat race 2008 editions. *SAT*, page 31, 2009.

[91] S. Subbarayan and D. K. Pradhan. NiVER: Non-increasing variable elimination resolution for preprocessing SAT instances. In *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT), Vancouver, BC, Canada*, pages 276–291, 2004.

[92] N. Tamura, A. Taga, S. Kitagawa, and M. Banbara. Compiling finite linear CSP into SAT. *Constraints*, 14(2):254–272, 2009.

[93] M. Velev and R. Bryant. Effective use of Boolean Satisfiability procedures in the formal verification of superscalar and VLIW microprocessors. In *Proceedings of the 38th annual Design Automation Conference, (DAC), Las Vegas, NV, USA*, pages 226–231, 2001.

[94] Y. Vizel, G. Weissenbacher, and S. Malik. Boolean Satisfiability solvers and their applications in model checking. *Proceedings of the IEEE*, 103(11):2021–2035, 2015.

[95] E. Yangin. Variable elimination algorithms for Maximum Satisfiability. Master's thesis, Universitat Autònoma de Barcelona, Barcelona, Spain, 2023.

[96] H. Zhang and J. Hsiang. Solving open quasigroup problems by propositional reasoning. In *Proceedings of the International Computer Symposium, (ICS) Hsinchu, Taiwan, December 12- 15*, 1994.

[97] H. Zhang, D. Li, and H. Shen. A SAT based scheduler for tournament schedules. In *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT), Vancouver, BC, Canada*, pages 10–13, 2004.

[98] L. Zhang and F. Bacchus. MAXSAT heuristics for cost optimal planning. In *Proceedings of the 26th Conference on Artificial Intelligence (AAAI), Toronto, Ontario, Canada*, pages 1846–1852, 2012.

[99] L. Zhang and S. Malik. The quest for efficient Boolean Satisfiability solvers. In *Proceedinds of the 18th International Conference on Automated Deduction (CADE), Copenhagen, Denmark*, pages 17–36, 2002.

[100] J. Zhu, A. Salhotra, C. R. Meinecke, P. Surendiran, R. Lyttleton, D. Reuter, H. Kugler, S. Diez, A. Månsson, H. Linke, and T. Korten. Solving the 3-SAT problem using network-based biocomputation. *Advanced Intelligent Systems*, 4:2200202, 2022.

Alphabetical Index

□-rule, 14
α-rule, 14
β-rule, 14
¬-rule, 14

Assignment, 8

Backward subsumption, 20
Boolean propositional logic, 7

Clause, 9
Compensation clause, 21
Compensation stack, 34
Conjunctive Normal Form, 9

formula
    hard, 9
    soft, 9
    weighted, 8
Forward subsumption, 20

Literal, 9

MaxSAT, 8
MaxSAT resolution, 20
MILP encoding, 79
MinSAT, 8
Model, 8
MSElim, 23

Occurrence lists, 28
One-literal rule, 19

propositional formula, 7
propositional logic
    Boolean, 7
    Regular, 11
    Signed, 10

Refutation, 13
Regular MaxSAT, 12
    Resolution, 49
    tableau calculus, 61
    Variable Elimination, 56
Regular MinSAT, 12
Regular SAT, 12
Resolution, 13
Resolvent, 13

SAT, 8
saturation
    MaxSAT, 23
    SAT, 18
Self-subsuming resolution, 20
Signed MaxSAT, 11
    Tableau calculus, 93
Signed MinSAT, 11
    Tableau calculus, 102
Signed SAT, 11
Subsumption, 20

Tableau calculus
    Clausal, 77
Tableau method, 13
Tautology, 8
Two-watched literal, 20

Uniform notation, 13
Unit Clause rule, 19

Unit Propagation, 19

Watch list, 20
Weighted MaxSAT, 8
Weighted MinSAT, 8
Weighted Partial MaxSAT, 9
Weighted Partial MinSAT, 9
Weighted Partial Regular MaxSAT, 12
    tableau calculus, 64
Weighted Partial Regular MinSAT, 12
    Tableau calculus, 70
Weighted Partial Signed MaxSAT, 11
    Tableau calculus, 95
Weighted Partial Signed MinSAT, 11
    Tableau calculus, 103
Weighted Regular MaxSAT
    Resolution, 58