Technische Universität Dresden

Master's Thesis

Unsatisfiability Proofs in SAT Solving with Parity Reasoning

Author:
Adrián Rebola-Pardo

Supervisors:
Prof. Dr. habil. Steffen Hölldobler
Tobias Philipp
(TU Dresden)

22nd November 2015

Abstract

SAT solvers have become very efficient in the last two decades due to the use of many simplification techniques and carefully designed data structures. The subsequent increase in code complexity led to reliability issues, and measures to certify results reported by SAT solvers were implemented. State-of-the-art SAT solvers are able to produce unsatisfiability proofs when confronted with an unsatisfiable instance. However, it was not known how to generate unsatisfiability proofs for a few very proficient techniques, and this situation forces these techniques to be disabled when unsatisfiability proofs are required. One such technique is parity reasoning, which is essential for the application of SAT solvers to cryptography.

In this Thesis, we solve the problem of generating unsatisfiability proofs for SAT solvers with integrated parity reasoning. In doing so, some theoretical contributions are proposed, including a generalized framework for proof systems that is able to model the DRAT proof standard used by state-of-the-art solvers. DRAT proofs present some unusual features; we show that they can nevertheless be modeled within an unified framework for proof systems. This framework allows composition operations on derivations, as well as the development of a notion of translation between proof systems. Our methods are based in translating execution steps in parity reasoning procedures, regarded as inferences in a proof system, into DRAT proof fragments. Finally, several methods to ensure the correctness of our approach and to reduce the size of generated proofs are presented.

Note for the revised version

This document is a revised version of the original Master's Thesis defended by the author in September 2015. Both versions differ exclusively in minor typographical and expressive details. The original Thesis is available online[1].

[1] `https://dl.dropboxusercontent.com/u/74942277/Papers/MasterThesis.pdf`

Acknowledgments

> It's very hard to talk quantum using a language originally designed to tell other monkeys where the ripe fruit is. [Pra02]
>
> *Sir Terry Pratchett (1948–2015)*

This Thesis is the product of many hours of enriching discussion with Tobias Philipp. It would be difficult to overemphasize how crucial his thoughtful guidance and infinite patience have been to bring this work to a successful end. I am obliged for every small and large knowledge he has provided me. I want to thank Steffen Hölldobler for giving me absolute freedom to develop the work presented in this Thesis, and for his kind support in my research. He and the rest of the faculty of the European Master in Computational Logic must be acknowledged, for they have taught me more than I ever expected when I enrolled in this program. I give thanks as well to the reviewers who agreed to read this Thesis.

Although they did not contribute directly to this Thesis, I would like to thank Peter Baumgartner and Luis Merino. Their support and advice in previous research have been somehow reflected here. I am grateful as well to Paola Bruscoli and Alessio Guglielmi. Many the ideas developed here, especially regarding proof systems, found an inspiration in their courses in the ESSLLI Summer School. This work was supported by an Erasmus Mundus grant awarded by the Education, Audiovisual and Culture Executive Agency in the European Union and the European Master's program in Computational Logic at TU Dresden; I am thankful for this unvaluable opportunity.

And last, but not least, I would like to thank my friends and family. If I can keep climbing onto the shoulders of giants is because they stand by my side.

Contents

Chapter 1

Introduction

SAT solving [BHvMW09] is a discipline in the intersection of logic, computer science and computer engineering that studies practical solutions to the *Boolean satisfiability problem*. The Boolean satisfiability problem involves a function that admits a vector of Boolean values 0 or 1, and returns one Boolean value, and the question whether there is an argument that makes this function to return 1.

A *clause* can be regarded as a function that returns value 1 if and only if some of the arguments take a predefined value 0 or 1. Clauses by themselves are not very expressive. However, for a choice of clauses, one may consider functions that return 1 if and only if all chosen clauses return 1. Such functions are called *clause normal form* (CNF) formulae. Most approaches to SAT solving admit only CNF formulae as input. In practice, this is not a severe restriction: any Boolean function with a finite number of arguments can be represented as a CNF formula.

However, it is possible that many clauses are necessary to represent some functions. For example, consider a function that returns 1 if and only if the number of input arguments with value 1 is even. These functions are called *parity constraints*. Parity constraints are particularly difficult to express as CNF formulae, as this requires an exponential number of clauses in the number of arguments of the parity constraint [GK13].

Modern SAT solvers operate directly over clauses [SLM09]. As a very rough approximation, SAT solvers try to find an argument making the function return 1 (in technical parlance, a satisfying *interpretation*) by guessing values for some variables in the argument. When all variables of a clause but one have been assigned a value, yet the clause cannot be guaranteed to return 1 (to become *satisfied*), the remaining variable can then be assigned to the unique value that would ensure this (the variable is *propagated*). If the solver finds a clause that is impossible to satisfy with the current assignments, it may undo some of them. To guarantee that the solver does not revisit the same assignment choice, a new clause can be added to the CNF formula (the clause is *learned*), constructed in such way that propagation itself prevents this situation. If the solver is finally able to learn a clause with no arguments (the *empty or unsatisfiable clause*), then the question whether a satisfying interpretation exists is answered negatively. If the solver finds an assignment satisfying every clause, then it is answered positively.

The high number of clauses required to represent parity constraints hinders the solver's ability to provide a result in reasonable time. Further problems arise from the use of CNF representations, called *encodings*, of parity constraints. On one hand, reasoning methods used by classical SAT solvers experience complexity blow ups in their presence [SNC09]. On the other hand, sets of parity constraints, called *affine formulae*, present problems regarding propagation properties. In particular, no CNF representation can guarantee that variables whose value under current guesses is forced (*entailed literals*) will eventually be propagated [GK13].

An interesting property is that reasoning on parity constraints is hard for classical SAT solvers [Urq87], but nonetheless efficient, complementary methods to deal with parity constraints exist. In this Thesis we

are particularly interested in *Gauss-Jordan elimination* [SNC09, Soo12], which is intimately related to the homonymous procedure for linear algebra [Day07]. In fact, Gauss-Jordan elimination over parity constraints involves obtaining a traditional row-echelon form for systems of linear congruences modulo 2 representing parity constraints.

To improve the performance of SAT solvers over problems with encoded parity constraints, which typically arise in cryptography [CB07, MM00], SAT solvers can be coupled with Gauss-Jordan elimination procedures [SNC09, Man14b, Lai14, Heu14]. This represented a breakthrough when solving such instances, and nowadays different integration strategies are used in state-of-the-art solvers [Soo12, Li00, Li03, LJN11, LJN10, LJN12, LJN14a, LJN14b, BM00, HvM05, WvM98, Che09, HDvZvM04].

A huge increase in performance of SAT solvers during the last decades led SAT solvers to be used in industrial problems [Rin09, LM06, GHM$^+$12, KS92, BK95], since a large class of problems generally regarded as intractable, called *NP-complete* problems, can be efficiently reduced to SAT solving [GJ79]. This performance leap was achieved by incremental improvements and implementation of many techniques and heuristics, such as *formula simplification* [EB05], *clause removal* [AS09], *lazy data structures* [MMZ$^+$01], *branching heuristics* [MMZ$^+$01, PD07], *restarts* [GSK98], *symmetry breaking* [ARMS03], efficient encodings for *pseudo-Boolean constraints* [ES06, MPS14] and advanced *learning schemata* [HS09, BKS04, MMZM01]. Consequently, current solvers present remarkably complex architectures. As expectable, both coding and logical errors became common, but debugging a SAT solver poses problems beyond the intricacy of the code. A false positive answer, that is, an unsatisfiable formula which is reported as satisfiable, is easily detected: the assignment generated by the solver can be checked against the input CNF formula to verify the answer.

False negatives are more problematic, since no easily verifiable witness is provided by the solving procedure, and it is an open problem whether this can be even be done in a tractable way [Pap07]. Witnesses called *unsatisfiability proofs* were developed to bridge the gap between false positives and false negatives, and increase reliability on modern SAT solvers. Unsatisfiability proofs are generally large and hard to check, but some advances allowed to palliate this issue. State-of-the-art proof formats, such as those based on *asymmetric tautologies* [HB15, HJB10, JHB12, WHJ14, HJW14], rely on the notion of learned clauses. Since learned clauses direct the search process during solving, a record of learned clauses serves as an unsatisfiability proof by checking if clauses were correctly learned. By recording only learned clauses instead of the full solving process, proof size decreases dramatically [HB15, HJW14].

The main issue in this Thesis is how to adapt this framework for unsatisfiability proof generation to solvers with a coupled parity reasoning module, which has been stated as an open question in the literature [HB15, SB06]. This is a non-trivial problem, because the internal functioning of SAT solvers and Gauss-Jordan elimination procedures are very different. Furthermore, proofs based on asymmetric tautologies are only adequate to record learned clauses, but there is not even a notion of clause in Gauss-Jordan elimination. Last, state-of-the-art techniques to integrate SAT solvers and parity reasoning modules interleave their execution in such a way that clauses learned by the SAT solver to the CNF formula are no longer asymmetric tautologies.

1.1 Previous work

Several methods for generation of unsatisfiability proofs in resolution-like proof systems for techniques used in SAT solving appear in the literature. Matti Järvisalo, Marijn Heule and Armin Biere proposed a formal framework for unsatisfiability proof generation for formula simplification techniques [JHB12]. Their work was subsequently extended for a large number of specific techniques by Norbert Manthey and Tobias Philipp [MP14]. Carsten Sinz, Armin Biere and Toni Jussila proposed methods to translate formula simplification using binary decision diagrams into extended resolution proofs [SB06, JSB06]. Last, recent work by Marijn Heule, Warren Hunt and Nathan Wetzler introduces DRAT proofs for symmetry breaking

techniques [SHvM09].

The issue of generating DRAT unsatisfiability proofs for parity reasoning procedures has been tackled obliquely in the literature. Cheng-Shen Han and Jie-Hong Jiang proposed proofs combining resolution and addition steps in the context of interpolant generation [HJ12]. Although a modification of the presented techniques to allow proof generation is not difficult, the generated proof would be expressed in a proof system allowing both CNF and affine formulae, as well as additional inference rules for parity constraints. Checking such proofs would imply developing new efficient proof verification methods, which is highly non-trivial [HJW14, WHJ14].

Both the resolution proof system underlying SAT solvers [PD11] and the state-of-the-art DRAT proof system [WHJ13] operate in a very different way to proof systems underlying some techniques applied in current SAT solvers (e.g. *binary decision diagrams* [JS04, GGW$^+$03], *cardinality resolution* [SS06, RM09] and *symmetry breaking* [Sak09, ARMS03, SHvM09]). Combining many different proofs systems would increase the code complexity of the proof checker, making it harder not only to develop, but also to certify through software verification. This would eventually give rise to the same correctness concerns that made proof generation necessary in the first place.

Another possible method to obtain unsatisfiability proofs when parity reasoning is involved uses *binary decision diagrams* (BDDs) [Bry86]. BDDs have been used to encode pseudo-Boolean constraints [BBR06, ES06] and as solving procedures integrated in SAT solvers [JS04, GGW$^+$03]. Intutively, BDDs are labeled acyclic directed graphs that represent propositional formulae. There exist algorithms [Bry86] that compute a BDD representing a propositional formula in a structurally recursive way. BDDs can be brought to a normal form called *reduced ordered BDDs (ROBDDs)* where satisfiability checking is immediate. In [SB06, JSB06], Carsten Sinz and Armin Biere propose a method to generate extended resolution derivations for encoded ROBDDs. They suggest the use of a similar method for the XOR connective to generate extended resolution translations of addition inferences in a XOR derivation. This method would slightly resemble the linear translation proposed in Chapter 7, but to the best of our knowledge it has not been developed beyond a few remarks in the literature. In fact, one of the authors mentioned reciently the problem as still unsolved [HB15].

Last, previous research by Laitinen, Junttila and Niemelä [LJN13] from a proof complexity perspective involves resolution proofs for some restricted versions of parity reasoning such as *equivalence reasoning*. Their work is directed towards complexity issues, focusing on what restrictions of parity reasoning might be actually beneficial to the performance of a SAT solver. Their theoretical methods are based on adding redundant parity constraints to improve unit propagation. Proofs of their results include techniques vaguely similar to those presented here. However, it is unclear how to turn them into actual unsatisfiability proofs; rather, their work tries to gain some insight on how to obtain shorter resolution proofs by adding redundant clauses to the premises.

1.2 Contributions

The solution of the problem of generating unsatisfiability proofs for SAT solvers with integrated parity reasoning involves the development of a full formal framework for proofs. By introducing a novel definition of proof system, we are able to model not only the notion of asymmetric tautologies, but also more general proof systems. A generalization of asymmetric tautologies, called *delete resolution asymmetric tautologies* (DRAT), is the current standard for unsatisfiability proofs in SAT solving. However, DRAT proofs are rather difficult to handle, due to significant differences with classical proof systems. For one, clauses derived by DRAT are not necessarily satisfied by all interpretations satisfying the original CNF formula, which is a property generally required in structural proof systems. Unifying DRAT in a common framework with other proof systems is an open problem, and generally ad hoc formalizations are used.

The framework for proof systems proposed in this Thesis formalizes DRAT proofs by allowing arbitrary *consequence relations* instead of the entailment relation. This allows the formalization of proof systems over constraints other than clauses and CNF formulae. In particular, we define the *XOR proof system* underlying Gauss-Jordan elimination. Once the execution of Gauss-Jordan elimination has been formalized as a proof, the problem of adapting the execution of Gauss-Jordan elimination to the DRAT proof system can be understood as a *translation* between proof systems. Furthermore, another proof system called *extended XOR* is proposed as an analogue of the extended resolution proof system for affine formulae. This allows to use a widespread technique from SAT solving, namely the use of *Tseitin variables* to obtain bounded-size constraints [Pre09], in the setting of parity constraints.

As a minor contribution, we introduce a generalized setting for *conflict analysis* and clause learning in SAT solving to model the use of diverse *clause learning schemata*, especially those used in solvers with integrated parity reasoning. Combined with the formal framework for proof systems, this allows the sound generation of proofs for general clause learning schemata in a modular way, such that only partial proofs need to be provided to a general generation procedure. This reduces our original problem of generating proofs for a whole family of solving procedures integrating SAT solving and parity reasoning to the simpler problem of obtaining translations of XOR proofs into DRAT proofs.

Obtaining such translations is the core of this Thesis. Using the formal framework for proof systems, we are able to generate two different methods for translation. The first one is called the *direct translation*, and involves translating every inference step in the XOR proof system into a derivation in the RAT system. Direct translations have a strong drawback in their exponential worst-case length. A second method is developed, called the *linear translation*. Linear translations involve *splitting* large parity constraints into smaller ones through the use of Tseitin variables. XOR inference steps over the larger parity constraints are first translated into *intermediate derivations* over the splitted parity constraints. Then, these inference steps are translated again using the direct translation. Since the obtained proof has premises and conclusions different from the encodings of the original parity constraints, *prefix* and *suffix* proofs must be appended to bridge the gap between them.

The replacement of arbitrary parity constraints in the original XOR proof by bounded-size parity constraints in the intermediate proof before direct translation is applied is the key to achieve much shorter DRAT proofs. We find theoretical results that show an exponential separation between direct and linear translations. However, this separation is asymptotic and therefore in practical situations direct translations may still be shorter than linear translations. Experiments suggest that this depends on the maximum value of a certain measure along the original XOR derivation. Hence, in practice both methods are useful. Nevertheless, tight length bounds are provided along the construction of both direct and linear translations, which allow to compute their lengths before they are computed, so as to decide which one to use.

1.3 Structure

This Thesis is structured in two parts. Part I is concerned with the development of useful formal frameworks that help reducing the general problem of generating unsatisfiability proofs for SAT solvers with parity reasoning modules to the problem of finding translations of proofs between XOR and DRAT proof systems. In Chapter 2, the notation used throughout our work is introduced, and the basics of CNF and affine formulae are presented. Chapter 3 presents our novel framework for proof systems, as well as the four proof systems that will be used in this Thesis. Furthermore, the fusion operations are introduced, and results to ensure sound proof composition are shown. Chapter 4 introduces the relevant aspects on SAT solving, with an emphasis on clause learning and unsatisfiability proof generation. The formal setting for conflict analysis and clause learning is introduced, and then instantiated for the case of classical CDCL SAT solving. Issues regarding parity reasoning are presented in Chapter 5. Both parity reasoning by itself using

Gauss-Jordan elimination, and its application to SAT solving through encodings and the XOR-CDCL SAT solving approach are discussed.

Part II deals with the main contribution in our work, that is, the translation of XOR proofs into DRAT proofs. Chapters 6 and 7 present detailed descriptions of direct and linear translations, respectively, as well as intuitive approximations to the ideas behind them. Together with correctness results, tight bounds for the lengths of both translations are provided. Chapter 8 discusses several issues in the application of our results to the generation of actual unsatisfiability proofs. In particular, theoretical and empirical comparisons between both encodings are presented, as well as results that allow to produce unsatisfiability proofs from our theoretical work. Finally, Chapter 9 reviews our achievements and results, and provides some insight on further issues that must be tackled in future work.

Part I

A formal framework for unsatisfiability proofs in SAT solving

Chapter 2

Preliminaries

This chapter is devoted to preliminaries in propositional logic. Section 2.1 introduces the main syntactical elements in our work. This includes mainly clauses and parity constraints, and different constructions and operations to combine them. Section 2.2 presents the semantics of these syntactical elements, and some results on how they relate to each other.

We first introduce our notation and present several basic results that will be used in this Thesis. Further notation is given in Figure 2.1. We use a specific notation for strings to avoid confusion between symbols and strings. *Strings* are finite sequences of elements. The string containing elements x_1, x_2, ..., x_n sequentially is denoted by $[x_1, x_2, \ldots, x_n]$. The empty string is then denoted by $[]$. The *concatenation* of two strings $u = [x_1, \ldots, x_n]$ and $v = [y_1, \ldots, y_m]$ is defined as $[x_1, \ldots, x_n, y_1, \ldots, y_m]$, and denoted by $u : v$.

The set of non-negative integers is denoted by $\mathbb{N}$; in particular, $0 \in \mathbb{N}$. Two integers x, y are said to be *equivalent modulo* 2, denoted by $x \approx y$ whenever they are both even or both odd. The following result from modular arithmetic will be used in this Thesis:

Proposition 2.1 (Congruence equivalence):
Let x, y, z, t be integers. Then, the non-congruence $x + y \not\approx z + t$ holds if and only if exactly one among $x \approx z$ and $y \approx t$ holds.

Proof. Straightforward to check by case exhaustion. □

The *symmetric difference* $S \triangle T$ of two sets S and T is the set $(S \setminus Y) \cup (Y \setminus S)$. The *power set* of a set S is denoted by $\mathcal{P}(S)$. Given two sets S and Y, *inclusion* of S in Y is denoted by $S \subseteq Y$, whereas *proper inclusion* of S in Y is denoted by $S \subset Y$. We sometimes refer to an *enumeration* $\{x_1, \ldots, x_n\}$ of a finite set S, which means that $S = \{x_1, \ldots, x_n\}$ and furthermore $s_i \neq s_j$ for every $0 \leq i < j \leq n$. If $f \colon S \longrightarrow S$ is a function, then we define

$$f^{0)}(x) = x \qquad\qquad f^{i+1)}(x) = f(f^{i)}(x))$$

for all $x \in S$ and $i \in \mathbb{N}$. That is, $f^{i)}$ represents the composition of f with itself i times. An interesting further result computes the number of parts of a set with cardinality of a specific parity:

Proposition 2.2 (Number of parts with a fixed parity cardinality):
Let S be a finite set. Let x be an integer and $S' = \{T \subseteq S : |T| \approx x\}$. Then, the cardinality of S' is given by:

$$|S'| = 2^{|S|-1}, \quad \text{if } S \neq \emptyset \qquad |S'| = 1, \quad \text{if } S = \emptyset \text{ and } x \approx 0 \qquad |S'| = 0, \quad \text{if } S = \emptyset \text{ and } x \approx 1$$

Proof. The cases for $S = \emptyset$ are straightforward from observing that $\mathcal{P}(\emptyset) = \{\emptyset\}$. We now show the case $S \neq \emptyset$.

Concrete propositional variables	p
Propositional variables	a, b, c
Splitting variables for parity constraint X	u^X
Sets of propositional variables	U, V, W
Interpretations	I, J
Literals	l, k
Sets of literals	L, M
Clauses	C, D, E
CNF formulae	F, G, H
Parity constraints	X, Y, Z
Affine formulae	A, B
General sets	S, T
Integers	$i, j, p, q, r, s, t, u, v, w, x, y, z$
Bijections	Φ, Ψ
Partial assignments	ν
Pseudoliterals	λ, μ
Implication graphs	K
Shifts on implication graphs	σ
General proof systems	R
Strings of CNF formulae	α, β, γ
Strings of affine formulae	φ, ψ, χ
DRAT representation	ε, η, ζ

Figure 2.1: Variables used to represent several objects. Unless stated otherwise, these symbols are used consistently along this Thesis, possibly with subindices, primes and other modifying features.

Let us enumerate $S = \{z_1, \ldots, z_n\}$, where $n > 0$, and define the set $S'' = \mathcal{P}(\{z_1, \ldots, z_{n-1}\})$. Consider now the well-defined mappings:

$$\Phi \colon S' \longrightarrow S'' : \Phi(T) = T \cap \{z_1, \ldots, z_{n-1}\} \qquad \Psi \colon S'' \longrightarrow S' : \Psi(T) = \begin{cases} T & \text{if } |T| \approx x \\ T \cup \{z_n\} & \text{if } |T| \not\approx x \end{cases}$$

Let $T \in S''$. If $|T| \approx x$, then $\Psi(T) = T \subseteq \{z_1, \ldots, z_{n-1}\}$, so we obtain $(\Phi \circ \Psi)(T) = T$. Similarly, if $|T| \not\approx x$, then since $z_n \notin T$ we obtain $(\Phi \circ \Psi)(T) = (T \cup \{z_n\}) \setminus \{z_n\} = T$. Hence, the composition $\Phi \circ \Psi$ is the identity mapping over S''.

Let now $T \in S'$. If $z_n \in T$, then $|\Phi(T)| = |T| - 1 \not\approx x$, so we obtain $(\Psi \circ \Phi)(T) = (T \setminus \{z_n\}) \cup \{z_n\} = T$. Otherwise, $z_n \notin T$ and we have that $|\Phi(T)| = |T| \approx x$, so $(\Psi \circ \Phi)(T) = T$. Therefore, the composition $\Psi \circ \Phi$ is the identity mapping over S'.

Thus, Ψ and Φ are mutually inverse bijections, and in particular $|S'| = |S''| = 2^{n-1}$. □

When dealing with bounds, the following result on partial sums of geometric series will be helpful:

Proposition 2.3 (Partial sum of a geometric series):
Let $r, s \in \mathbb{N}$ with $r < s$. Then, $\sum_{i=r}^{s} 2^i = 2^{s+1} - 2^r$.

Proof. Consider the partial sum of a geometric series for $a \neq 1$ and $n \in \mathbb{N}$, given by:

$$\sum_{i=0}^{n} a^i = \frac{1 - a^{n+1}}{1 - a}$$

In the case when $a = 2$, we obtain $\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$. Hence:

$$\sum_{i=r}^{s} 2^i = \sum_{i=0}^{s} 2^i - \sum_{i=0}^{r-1} 2^i = (2^{s+1} - 1) - (2^r - 1) = 2^{s+1} - 2^r \qquad \square$$

2.1 Syntax of propositional logic

SAT solvers operate in the framework of *propositional logic*. Propositional logic is usually introduced by a definition of *propositional formulae* and their semantics, followed by the notion of *clausal normal form* (CNF) formulae, which are presented as a special fragment of propositional formulae. In this work, we define the fundamentals of propositional logic relying exclusively on CNF formulae, and we introduce *parity constraints* as additional syntactical elements. In doing so, we do not lose or gain any expresivity for our language: propositional and CNF formulae have the same expressive power, since every CNF formula can be regarded as a propositional formula and every propositional formula can be transformed into a semantically equivalent CNF formula [vD94, Theorem 1.3.9].

We consider a fixed, countably infinite set of *variables* $\mathbf{Var} = \{\mathsf{p}_i : i \in \mathbb{N}\}$. We furthermore assume a symbol $\top$ not occurring in **Var**. A *pseudovariable* is either a variable or the $\top$ symbol. The union $\mathbf{Var} \cup \{\top\}$ is denoted by $\mathbf{Var}^\star$.

Definition 2.4 (Literals):
A literal *is either a variable $a \in \mathbf{Var}$, or its* negation, *denoted by $\neg a$. In the first case, the literal is said to be* positive; *otherwise, it is* negative. *We denote the set of literals as* **Lit**. *Furthermore, the following operations are defined on literals:*

- *The* complement *$\overline{a}$ of a positive literal a is $\neg a$; the complement $\overline{\neg a}$ of a negative literal $\neg a$ is a.*
- *The* underlying variable $\mathrm{var}(a)$ *of a positive literal a is a itself; the underlying variable* $\mathrm{var}(\neg a)$ *of a negative literal $\neg a$ is a.*
- *The* set of atoms $\mathrm{atoms}(l)$ *of a literal l is $\{l, \overline{l}\}$.*

Two literals l_1, l_2 are said to be complementary *whenever $\overline{l_1} = l_2$ (or equivalently $\overline{l_2} = l_1$). A set of literals L is said to be* consistent *if, for every literal l, if $l \in L$ then $\overline{l} \notin L$; it is otherwise called* inconsistent.

Example 2.5 (Literals):
Consider the literals p_1 and $\neg\mathsf{p}_2$. Their values for the operations in Definition 2.4 are given in the following table:

	l	p_1	$\neg\mathsf{p}_2$
Complement	$\overline{l}$	$\neg\mathsf{p}_1$	p_2
Underlying variable	$\mathrm{var}(l)$	p_1	p_2
Atoms	$\mathrm{atoms}(l)$	$\{\mathsf{p}_1, \neg\mathsf{p}_1\}$	$\{\mathsf{p}_2, \neg\mathsf{p}_2\}$

Complementation of a literal switches the negation sign before the underlying variable, given by the variable inside the literal. The set of atoms of a literal is given by all possible literals with the same underlying variable. Examples for consistency of sets of literals are provided in Example 2.9. ■

Although the semantics of propositional logic are introduced later, the underlying notions are hinted here. *Boolean interpretations* are the central semantic notion, and map variables to one of the *truth values* 0 or 1. The negation symbol flips the truth assignment of a variable, i.e. if a has truth value x, then $\neg a$ has truth value $1 - x$. A variable or literal is *satisfied* by an interpretation whenever its truth assignment is 1. In the following, we present two kinds of semantic elements. *Clauses* are constraints forcing an interpretation

to satisfy at least one literal occurring in the clause. In contrast, *parity constraints* (also called xor-clauses in the literature [GK13]) force the number of satisfied variables occurring in it to be either even or odd. Conjunctions of clauses are called *CNF formulae*, and conjunctions of parity constraints are called *affine formulae*, and are represented simply as sets of the corresponding constraints.

Definition 2.6 (Clauses):
A clause *is an expression of the form* $\mathsf{or}(L)$*, where* L *is a finite, consistent set of literals. We denote the set of clauses by* $\mathbf{Cla}$*. Whenever the set inside* $\mathsf{or}(\ )$ *is specified with set brackets* $\{\ \}$*, we will drop the brackets, e.g.* $\mathsf{or}(l_1, \ldots, l_k)$ *and also* $\mathsf{or}(l \in L : l \notin M)$*. The following operations are defined on clauses:*

- *The* set of literals $\mathrm{lits}(C)$ *of a clause* $C = \mathsf{or}(L)$ *is* L*.*
- *The* set of variables $\mathrm{vars}(C)$ *of a clause* C *is* $\{\mathrm{var}(l) : l \in \mathrm{lits}(C)\}$*.*
- *The* set of atoms $\mathrm{atoms}(C)$ *of a clause* C *is* $\bigcup_{l \in \mathrm{lits}(C)} \mathrm{atoms}(l)$*.*
- *The* size $|C|$ *of a clause* C *is the number of elements in* $\mathrm{lits}(C)$*.*
- *Two clauses* C, D *are* disjunctable *if* $\mathrm{lits}(C) \cup \mathrm{lits}(D)$ *is consistent; in that case,* $\mathsf{or}(\mathrm{lits}(C) \cup \mathrm{lits}(D))$ *is a clause, called their* disjunction *and denoted by* $C \vee D$*.*
- *Given a finite, consistent set* L *of literals,* $\mathrm{nand}(L)$ *is defined as the clause* $\mathsf{or}(\neg l : l \in L)$*.*

A clause C subsumes *a clause* D *if* $\mathrm{lits}(C) \subseteq \mathrm{lits}(D)$*. Furthermore,* C *is said to be a* unit *clause whenever* $\mathrm{lits}(C)$ *contains exactly one literal.*

The most important operation on clauses this Thesis is concerned with is *resolution* and one of its extensions, called *linear resolution*.

Definition 2.7 (Resolvents):
Consider two clauses C*,* D *and a literal* l*. We say that* C *is* resolvable *with* D *upon* l *whenever the following hold:*

i) $l \in \mathrm{lits}(C)$ *and* $\overline{l} \in \mathrm{lits}(D)$*.*

ii) For all $k \in \mathrm{lits}(C)$*, if* $k \neq l$ *then* $\overline{k} \notin \mathrm{lits}(D)$*.*

iii) For all $k \in \mathrm{lits}(D)$*, if* $k \neq \overline{l}$ *then* $\overline{k} \notin \mathrm{lits}(C)$*.*

Whenever these hold, the set of literals $L = (\mathrm{lits}(C) \setminus \{l\}) \cup (\mathrm{lits}(D) \setminus \{\overline{l}\})$ *is consistent, and thus we can define the* resolvent *of* C *with* D *upon* l *as the clause* $C \otimes_l D = \mathsf{or}(L)$*.*

Observe that, whenever C *is resolvable with* D *upon a literal* l*, then they are not resolvable upon any other literal. This justifies that sometimes we simply say that* C *is resolvable with* D*, and write the resolvent as* $C \otimes D$*. In general, the resolution operation is not associative. We write iterated resolutions in a left-associative way, e.g.* $C_1 \otimes C_2 \otimes C_3$ *stands for* $(C_1 \otimes C_2) \otimes C_3$*.*

A clause E *is said to be a* resolvent *in a set of clauses* F *whenever there are clauses* $C, D \in F$ *such that* C *is resolvable with* D *with resolvent* $C \otimes D = E$*.*

Definition 2.8 (Linear resolution):
Let F *be a set of clauses. A* linear resolution *in* F *is a string of clauses* $[C_0, \ldots, C_n]$ *with the following properties:*

i) $C_0 \in F$*.*

ii) For every $1 \leq i \leq n$*, there is a clause* $D_i \in F$ *such that* C_{i-1} *is resolvable with* D_i *with* $C_{i-1} \otimes D_i = C_i$*.*

A clause C *is called a* linear resolvent *in* F *if there is a linear resolution* $[C_0, \ldots, C_n]$ *in* F *with* $C_n = C$*.*

Example 2.9 (Clauses and resolvents):
Consider the sets of literals:

$$L_1 = \{p_1, \neg p_2, \neg p_3\} \qquad L_3 = \{p_0, \neg p_2, p_5\}$$
$$L_2 = \{\overline{\neg p_i} : i \leq 3\} = \{p_0, p_1, p_2, p_3\} \qquad L_4 = \{p_0, p_2, \neg p_2\}$$

L_1, L_2 and L_3 are consistent, whereas L_4 is not because it contains the pair of complementary literals $p_2, \neg p_2$. Furthermore, we have $\mathrm{pol}(L_1) = \mathrm{pol}(L_2) = 0$ whereas $\mathrm{pol}(L_3) = \mathrm{pol}(L_4) = 0$. Now consider the clauses:

$$\mathsf{or}(L_1) = \mathsf{or}(p_1, \neg p_2, \neg p_3)$$
$$\mathsf{or}(L_2) = \mathsf{or}(\overline{\neg p_i} : i \leq 3) = \mathsf{or}(p_0, p_1, p_2, p_3)$$
$$\mathsf{or}(L_3) = \mathsf{or}(p_0, \neg p_2, p_5)$$

Note that $\mathsf{or}(L_4)$ is not defined because L_4 is inconsistent. The values of $\mathsf{or}(L_1)$ and $\mathsf{or}(L_3)$ for the operations in Definition 2.6 are given in the following table:

	C	$\mathsf{or}(p_1, \neg p_2, \neg p_3)$	$\mathsf{or}(p_0, \neg p_2, p_5)$
Set of variables	$\mathrm{vars}(C)$	$\{p_1, p_2, p_3\}$	$\{p_0, p_2, p_5\}$
Set of literals	$\mathrm{lits}(C)$	$\{p_1, \neg p_2, \neg p_3\}$	$\{p_0, \neg p_2, p_5\}$
Set of atoms	$\mathrm{atoms}(C)$	$\{p_1, \neg p_1, p_2, \neg p_2, p_3, \neg p_3\}$	$\{p_0, \neg p_0, p_2, \neg p_2, p_5, \neg p_5\}$
Size	$\lvert C\rvert$	3	3

The set of variables collects the underlying variables to all literals occurring in a clause. The set of literals is simply the set inside the $\mathsf{or}(\ )$ construct. The set of atoms is the set of variables together with their negations. Clauses are disjunctable whenever the union of their literals does not contain any pair of complementary literals. Clauses $\mathsf{or}(L_1)$ and $\mathsf{or}(L_2)$ are not disjunctable, since $\neg p_2 \in L_1$ and $p_2 \in L_2$. However, $\mathsf{or}(L_1)$ and $\mathsf{or}(L_3)$ are disjunctable with $\mathsf{or}(L_1) \vee \mathsf{or}(L_3) = \mathsf{or}(p_0, p_1, \neg p_2, p_3, \neg p_5)$.

Two clauses are resolvable upon a literal l whenever $l, \bar{l}$ is the only pair of complementary literals occurring in both clauses together. Clauses $\mathsf{or}(L_2)$ and $\mathsf{or}(L_1)$ are not resolvable upon p_2, because $p_3 \in \mathsf{or}(L_2)$ and $\neg p_3 \in \mathsf{or}(L_1)$ is another pair of complementary literals. However, $\mathsf{or}(L_2)$ and $\mathsf{or}(L_3)$ are resolvable upon p_2 with resolvent $\mathsf{or}(L_2) \otimes_{p_2} \mathsf{or}(L_3) = \mathsf{or}(p_0, p_1, p_3, p_5)$. In fact, one can build the following chain of resolutions:

$$\mathsf{or}(L_3) \otimes_{\neg p_2} \mathsf{or}(L_2) \otimes_{p_3} \mathsf{or}(L_1) = \mathsf{or}(p_0, \neg p_2, p_5) \otimes \mathsf{or}(p_0, p_1, p_2, p_3) \otimes \mathsf{or}(p_1, \neg p_2, \neg p_3) =$$
$$= \mathsf{or}(p_0, p_1, p_3, p_5) \otimes \mathsf{or}(p_1, \neg p_2, \neg p_3) = \mathsf{or}(p_0, p_1, \neg p_2, p_5)$$

Therefore, $\mathsf{or}(p_0, p_1, \neg p_2, p_5)$ is a linear resolvent in $\{\mathsf{or}(L_1), \mathsf{or}(L_2), \mathsf{or}(L_3)\}$ generated by the linear resolution:

$$[\mathsf{or}(p_0, \neg p_2, p_5)\ ,\ \mathsf{or}(p_0, p_1, p_3, p_5)\ ,\ \mathsf{or}(p_0, p_1, \neg p_2, p_5)]$$ ■

Definition 2.10 (CNF formulae):
A clause normal form (CNF) formula *is a finite set of clauses. The set of all CNF formulae is denoted by* **Cnf**. *The following operations are defined on CNF formulae:*

- *The* set of variables $\mathrm{vars}(F)$ *of a CNF formula F is* $\bigcup_{C \in F} \mathrm{vars}(C)$.
- *The* set of atoms $\mathrm{atoms}(F)$ *of a CNF formula F is* $\bigcup_{C \in F} \mathrm{atoms}(C)$.
- *The* size $|F|$ *of a CNF formula F is the number of clauses in F.*

Furthermore, we define the reduct *of a CNF formula F by a consistent set of literals L, as the CNF formula:*

$$F|_L = \{\mathsf{or}(l \in \mathrm{lits}(C) : \bar{l} \notin L) : C \in F \text{ and } \mathrm{lits}(C) \cap L = \emptyset\}$$

F is said to be unresolvable *if there is no pair of clauses $C, D \in F$ such that C is resolvable with D.*

Our work is concerned with a second type of constraints, called *parity constraints* [GK13]. Parity constraints are important for SAT solving because CNF formulae encoding parity constraints constitute hard SAT problems [Urq87]. Nevertheless, efficient algorithms such as *Gauss-Jordan elimination* exist for parity constraints. Integration of parity reasoning in SAT solving is a non-trivial issue and, due to the occurrence of encoded parity constraints in many cryptographic problems [MM00], much effort has been invested in research on these problems [Lai14, SNC09, Che09]. These issues are further tackled in Chapter 5.

Definition 2.11 (Parity constraints):
A parity constraint *is an expression of the form $\mathsf{par}(V)$, where V is a finite set of pseudovariables, i.e. elements of V are either variables or the $\top$ symbol. We denote the set of parity constraints by* **Par**.*We adopt the notational convention described for clauses to drop set brackets inside a $\mathsf{par}(\ )$ expression, i.e. we admit expressions like $\mathsf{par}(a_1, \ldots, a_k)$ and $\mathsf{or}(a \in V : a \notin W)$. The following operations are defined on parity constraints:*

- *The* set of pseudovariables $\mathrm{vars}^\star(X)$ *of a parity constraint $X = \mathsf{par}(V)$ is V.*
- *The* set of variables $\mathrm{vars}(X)$ *of a parity constraint X is* $\mathrm{vars}^\star(X) \setminus \{\top\}$.
- *The* set of atoms $\mathrm{atoms}(X)$ *of a parity constraint X is* $\bigcup_{a \in \mathrm{vars}(X)} \mathrm{atoms}(a)$.
- *The* polarity $\mathrm{pol}(X)$ *of a parity constraint X is* 1 *if* $\top \in \mathrm{vars}^\star(X)$*; and* 0 *otherwise.*
- *The* size $|X|$ *of a parity constraint X is the number of variables in* $\mathrm{vars}(X)$.
- *The* ω-measure *of two parity constraints X, Y is defined as* $\omega(X, Y) = |\mathrm{vars}(X) \cup \mathrm{vars}(Y)|$.
- *The* δ-measure *of two parity constraints X, Y is defined as* $\delta(X, Y) = |\mathrm{vars}(X) \triangle \mathrm{vars}(Y)|$.
- *The* parity constraint version *of a literal l, denoted by* $\mathrm{xor}(l)$*, is* $\mathsf{par}(\mathrm{var}(l), \top)$ *if l is a positive literal; and* $\mathsf{par}(\mathrm{var}(l))$ *if l is a negative literal.*
- *The* polarization $X_{(Y)}$ *of a parity constraint X by another parity constraint Y is a parity constraint with all variables from X and the polarity from Y. That is:*

$$X_{(Y)} = \mathsf{par}(\mathrm{vars}(X) \cup \{\top\}), \quad \text{if } \mathrm{pol}(Y) = 1 \qquad X_{(Y)} = \mathsf{par}(\mathrm{vars}(X)), \quad \text{if } \mathrm{pol}(Y) = 0$$

- *The* addition $X \oplus Y$ *of two parity constraints X, Y is defined as* $\mathsf{par}(\mathrm{vars}^\star(X) \triangle \mathrm{vars}^\star(Y))$.

Example 2.12 (Parity constraints):
Consider the parity constraints:

$$X_1 = \mathsf{par}(\mathsf{p}_1, \mathsf{p}_2, \mathsf{p}_3, \top) \qquad X_2 = \mathsf{par}(\mathsf{p}_2, \mathsf{p}_4, \top) \qquad X_3 = \mathsf{par}(\mathsf{p}_1, \mathsf{p}_4) \qquad X_4 = \mathsf{par}(\mathsf{p}_3, \top)$$

The values of X_1 and X_3 for the operations in Definition 2.11 are given in the following table:

	X	$\mathsf{par}(\mathsf{p}_1, \mathsf{p}_2, \mathsf{p}_3, \top)$	$\mathsf{par}(\mathsf{p}_1, \mathsf{p}_4)$
Set of pseudovariables	$\mathrm{vars}^\star(X)$	$\{\mathsf{p}_1, \mathsf{p}_2, \mathsf{p}_3, \top\}$	$\{\mathsf{p}_1, \mathsf{p}_4\}$
Set of variables	$\mathrm{vars}(X)$	$\{\mathsf{p}_1, \mathsf{p}_2, \mathsf{p}_3\}$	$\{\mathsf{p}_1, \mathsf{p}_4\}$
Set of atoms	$\mathrm{atoms}(X)$	$\{\mathsf{p}_1, \neg\mathsf{p}_1, \mathsf{p}_2, \neg\mathsf{p}_2, \mathsf{p}_3, \neg\mathsf{p}_3\}$	$\{\mathsf{p}_1, \neg\mathsf{p}_1, \mathsf{p}_4, \neg\mathsf{p}_4\}$
Polarity	$\mathrm{pol}(X)$	1	0
Size	$\lvert X\rvert$	3	2

The set of pseudovariables is simply the set inside the $\mathsf{par}(\ )$ construct. The set of variables collects all non-$\top$ symbols (i.e. variables) inside the $\mathsf{par}(\ )$ construct. The set of atoms is the set of variables together with their negations. Finally, the polarity of a parity constraint is 1 if the $\top$ symbol occurs inside the constraint, and 0 otherwise.

The polarization of a parity constraint X by a parity constraint Y yields a new parity constraint with all variables from X and the polarity of Y. For example, the polarization of X_1 by X_2 is $X_{1_{(X_2)}} = X_1$ because $\top$ occurs in both X_1 and X_2. However, the polarization of X_1 by X_3 is $X_{1_{(X_3)}} = \mathsf{par}(\mathsf{p}_1, \mathsf{p}_2, \mathsf{p}_3)$, because $\top$ does not occur in X_3.

Finally, we provide some examples of parity constraint addition:

$$X_1 \oplus X_2 = \mathsf{par}(\mathsf{p}_1, \mathsf{p}_3, \mathsf{p}_4) \qquad X_2 \oplus X_3 = \mathsf{par}(\mathsf{p}_1, \mathsf{p}_2, \top) \qquad X_1 \oplus X_4 = \mathsf{par}(\mathsf{p}_1, \mathsf{p}_2) \qquad \blacksquare$$

Definition 2.13 (Affine formulae):
An affine formula *is a finite set of parity constraints. The set of all affine formulae is denoted by* **Aff**. *The following operations are defined on affine formulae:*

- The set of variables $\mathrm{vars}(A)$ *of an affine formula* A *is* $\bigcup_{X \in A} \mathrm{vars}(A)$.
- The set of atoms $\mathrm{atoms}(A)$ *of an affine formula* A *is* $\bigcup_{X \in A} \mathrm{atoms}(A)$.
- *The* size $|A|$ *of an affine formula* A *is the number of parity constraints in* A.

2.2 Semantics of propositional logic

Our presentation so far includes syntactical definitions without a formal notion of truth or consequence. Semantics of propositional logic are based on objects called *Boolean interpretations*, together with a *satisfaction relation* for constraints, which determines whether a constraint is true under an interpretation. Later, consequence relations between constraints are introduced.

Definition 2.14 (Boolean interpretations):
The set of truth values $\{0, 1\}$ *is denoted by* $\mathbb{B}$. *A* Boolean interpretation, *or simply an interpretation, is a mapping* $I\colon \mathbf{Var} \longrightarrow \mathbb{B}$. *The* satisfaction relation *between Boolean interpretations and the constraints presented in Section 2.2 is defined as follows for a Boolean interpretation* I*:*

- I *satisfies* $\top$.
- I *satisfies a negative literal* $\neg a$ *if and only if* I *does not satisfy* a.
- I *satisfies a clause* C *if and only if* I *satisfies some literal* $l \in \mathrm{lits}(C)$.
- I *satisfies a CNF formula* F *if and only if* I *satisfies all clauses* $C \in F$.
- I *satisfies a parity constraint* X *if and only if* $\sum_{a \in \mathrm{vars}(X)} I(a) \approx \mathrm{pol}(X)$*; or equivalently, if the number of pseudovariables in* $\mathrm{vars}^\star(X)$ *satisfied by* I *is even.*
- I *satisfies an affine formula* A *if and only if* I *satisfies all parity constraints* $X \in A$.

We use the notation $I \models \vartheta$ *to denote that an interpretation* I *satisfies a constraint* θ*. Whenever* I *does not satisfy one of the constraints above, we say that* I falsifies *the constraint.*

Definition 2.15 (Flippings of an interpretation):
Given an interpretation I *and a set* V *of variables, the* flipping *of* V *on* I*, denoted by* I/V*, is an interpretation such that* $(I/V)(a) = \overline{I(a)}$ *for every variable* $a \in V$*, and* $(I/V)(a) = I(a)$ *otherwise.*

The semantics defined above allow further definitions on how the semantics of different constraints relate to each other. Here, "constraint" refers to either pseudovariables, literals, clauses, CNF formulae, parity constraints or affine formulae.

Definition 2.16 (Entailment and satisfiability):
Let ϑ_1, ϑ_2 be two constraints. We define the following properties:

- ϑ_1 *is* satisfiable *if there exists a Boolean interpretation satisfying ϑ_1; otherwise it is said to be* unsatisfiable.
- ϑ_1 *is* valid *if all Boolean interpretations satisfy ϑ_1.*
- ϑ_1 entails ϑ_2 *(symbolically $\vartheta_1 \models \vartheta_2$) if every Boolean interpretation satisfying ϑ_1 satisfies ϑ_2 as well.*
- ϑ_1 *is* semantically equivalent *to ϑ_2 (symbolically $\vartheta_1 \equiv \vartheta_2$) if, for every Boolean interpretation I, I satisfies ϑ_1 if and only if I satisfies ϑ_2; or, equivalently, if $\vartheta_1 \models \vartheta_2$ and $\vartheta_2 \models \vartheta_1$.*
- ϑ_1 *is* cosatisfiable *with ϑ_2 (symbolically $\vartheta_1 \models_{\mathsf{sat}} \vartheta_2$) if satisfiability of ϑ_1 implies satisfiability of ϑ_2.*
- ϑ_1 *is* equisatisfiable *to ϑ_2 (symbolically $\vartheta_1 \equiv_{\mathsf{sat}} \vartheta_2$) if they are both satisfiable or both unsatisfiable.*

Proposition 2.17 (Basic results for entailment and satisfiability):
Let ϑ_1, ϑ_2, ϑ_3 be constraints. The following hold:

i) If $\vartheta_1 \models \vartheta_2$ and $\vartheta_2 \models \vartheta_3$, then $\vartheta_1 \models \vartheta_3$. That is, $\models$ is transitive.

ii) If $\vartheta_1 \models_{\mathsf{sat}} \vartheta_2$ and $\vartheta_2 \models_{\mathsf{sat}} \vartheta_3$, then $\vartheta_1 \models_{\mathsf{sat}} \vartheta_3$. That is, $\models_{\mathsf{sat}}$ is transitive.

iii) If $\vartheta_1 \models \vartheta_2$, then $\vartheta_1 \models_{\mathsf{sat}} \vartheta_2$.

iv) $\vartheta_1 \equiv \vartheta_2$ holds if and only if both $\vartheta_1 \models \vartheta_2$ and $\vartheta_2 \models \vartheta_1$ hold.

v) $\vartheta_1 \equiv_{\mathsf{sat}} \vartheta_2$ holds if and only if both $\vartheta_1 \models_{\mathsf{sat}} \vartheta_2$ and $\vartheta_2 \models_{\mathsf{sat}} \vartheta_1$ hold.

Proof. Straightforward from Definition 2.16. □

The rest of this section is devoted to illustrate this notion by means of examples, and present a few basic results on the semantics defined above. Some of the results expounded below are very basic and can be found in many textbooks, although some notions may vary slightly.

Example 2.18 (Semantics of clauses and parity constraints):
Consider the Boolean interpretation I defined by $I(\mathsf{p}_i) = 1$ if $i \leq 3$ and $I(\mathsf{p}_i) = 0$ otherwise. Given the set $V = \{\mathsf{p}_2, \mathsf{p}_4\}$, the Boolean interpretation I/V is given by $I(\mathsf{p}_i) = 1$ if $i \in \{0, 1, 3, 4\}$ and $I(\mathsf{p}_i) = 0$ otherwise. Then, we have the following table of satisfied expressions:

Constraint ϑ	$I \models \vartheta$	$I/V \models \vartheta$
p_1	✓	✓
$\neg\mathsf{p}_3$		
$\overline{\neg\mathsf{p}_2}$	✓	
$\top$	✓	✓
$C_1 = \mathsf{or}(\neg\mathsf{p}_1, \mathsf{p}_2, \neg\mathsf{p}_3)$	✓	
$C_2 = \mathsf{or}(\neg\mathsf{p}_2, \neg\mathsf{p}_4)$	✓	✓
$C_1 \otimes_{\mathsf{p}_2} C_2 = \mathsf{or}(\neg\mathsf{p}_1, \neg\mathsf{p}_3, \neg\mathsf{p}_4)$	✓	
$\{C_1, C_2\}$	✓	
$X_1 = \mathsf{par}(\mathsf{p}_0, \mathsf{p}_1, \mathsf{p}_2)$		✓
$X_2 = \mathsf{par}(\mathsf{p}_1, \mathsf{p}_5, \top)$	✓	✓
$X_1 \oplus X_2 = \mathsf{par}(\mathsf{p}_0, \mathsf{p}_2, \mathsf{p}_5, \top)$		✓
$\{X_1, X_2\}$		✓

Observe that in the table above whenever an interpretation satisfies C_1 and C_2, it satisfies $C_1 \otimes_{\mathsf{p}_2} C_2$ as well. Similarly, if an interpretation satisfies X_1 and X_2, then it satisfies $X_1 \oplus X_2$, too. This is a general behavior, as shown by Propositions 2.19 and 2.21. ■

Proposition 2.19 (Basic results about clauses):
Let C, D be clauses, F, G, H be CNF formulae, l be a literal and I be a Boolean interpretation. The following hold:

i) *If C and D are disjunctable, then $I \models C \vee D$ if and only if either $I \models C$ or $I \models D$.*

ii) *If $C, D \in F$ and C is resolvable with D, then $F \models C \otimes D$.*

iii) *If C is a linear resolvent in F, then $F \models C$.*

iv) *If C subsumes D, then $I \models C$ implies $I \models D$.*

v) *The only unsatisfiable clause is $\mathsf{or}(\emptyset)$.*

vi) *If $F \subseteq G$, then $G \models F$.*

vii) *If $F \models G$ and $F \models H$, then $F \models G \cup H$.*

viii) $\mathsf{or}(l) \equiv l$.

Proof.

i) Straightforward from the definition of disjunction.

ii) Let l be a literal upon which C is resolvable with D. If I satisfies F, then in particular it satisfies both clauses C and D. Then, there are literals $k \in \mathrm{lits}(C)$ and $m \in \mathrm{lits}(D)$ satisfied by I. This implies that k and m cannot be complementary. In particular, at least one of them is neither l nor $\bar{l}$. Without loss of generality, assume it is k. Then, $k \in \mathrm{lits}(C \otimes_l D)$, so I satisfies $C \otimes_l D$.

iii) Straightforward from *(ii)* by induction on the length of the linear resolution.

iv) If I satisfies C, then there is a literal $l \in \mathrm{lits}(C) \subseteq \mathrm{lits}(D)$ satisfied by I.

v) A clause C is unsatisfiable if and only if no interpretation satisfies any literal in $\mathrm{lits}(C)$, which can only happen if $\mathrm{lits}(C) = \emptyset$.

vi) Straightforward from Definition 2.16.

vii) Straightforward. □

Claim *(ii)* in Proposition 2.19 shows that resolution can be used as a syntactical procedure to derive consequences. Resolution is of an utmost importance in SAT solving [KL99, PD11] and propositional logic in general. It provides a syntactical procedure to derive consequences, although as we will see in Example 3.9 not all consequences can be derived by computing resolvents.

Example 2.20 (Semantics of CNF formulae):
Consider the following clauses and CNF formulae:

$$
\begin{aligned}
C_1 &= \mathsf{or}(\mathsf{p}_1, \mathsf{p}_2, \mathsf{p}_3) \\
C_2 &= \mathsf{or}(\neg \mathsf{p}_1) \qquad\qquad & F &= \{C_1, C_2, C_3, C_4\} \\
C_3 &= \mathsf{or}(\mathsf{p}_1, \neg \mathsf{p}_2) & G &= \{C_2, C_3, C_4\} \\
C_4 &= \mathsf{or}(\mathsf{p}_2, \neg \mathsf{p}_3) & H &= \{C_2, C_3, C_4, C_5\} \\
C_5 &= \mathsf{or}(\mathsf{p}_1, \neg \mathsf{p}_3)
\end{aligned}
$$

The following hold:

- *F is unsatisfiable* if F was satisfiable, then we would have an interpretation I with $I \models F$. This means that I satisfies all C_1, C_2, C_3, C_4. Since I satisfies C_2, then $I \not\models \neg \mathsf{p}_1$. Now, I also satisfies the clause $C_3 = \mathsf{or}(\mathsf{p}_1, \neg \mathsf{p}_2)$, so $I \not\models \mathsf{p}_2$. The same argument applied to C_4 gives $I \not\models \mathsf{p}_3$, so we conclude that $I(\mathsf{p}_1) = I(\mathsf{p}_2) = I(\mathsf{p}_3) = 0$. But this in particular means that I does not satisfy C_1, which is a contradiction. Thus, there is no Boolean interpretation satisfying F, so F is unsatisfiable.
- *G is satisfiable* consider any interpretation I with $I(\mathsf{p}_1) = I(\mathsf{p}_2) = I(\mathsf{p}_3) = 0$. As seen above, I then satisfies G.
- *H is satisfiable* the same interpretation I above satisfies C_5, so I satisfies H. In fact, this is not surprising because C_3 is resolvable with C_4 upon $\neg \mathsf{p}_2$ with resolvent C_5, and by Proposition 2.19, every interpretation satisfying G must satisfy C_5 as well.
- $F \models_{\mathsf{sat}} G$ *and* $F \models G$ both are straightforward from unsatisfiability of F.
- $F \not\equiv_{\mathsf{sat}} G$ *but* $G \equiv_{\mathsf{sat}} H$ this is a consequence of F being unsatisfiable and G and H being satisfiable.
- $G \models H$ as explained above, a consequence of $G \models C_5$.
- $G \equiv H$ we have shown above that $G \models H$. Furthermore, $G \subseteq H$, so any interpretation satisfying H must satisfy G as well, so $H \models G$. ■

Proposition 2.21 (Basic results about parity constraints):
Let X, Y, Z be parity constraints, A, B, B' be affine formulae, V be a set of variables, l a literal and I be a Boolean interpretation. The following hold:

i) $I \models X \oplus Y$ *if and only if I satisfies either both X and Y, or none of them.*

ii) $X \oplus Y = Y \oplus Z$.

iii) $(X \oplus Y) \oplus Z = X \oplus (Y \oplus Z)$.

iv) $X \oplus \mathsf{par}(\emptyset) = X$.

v) $I \models X_{(Y)}$ *if and only if* $\sum_{a \in \mathrm{vars}(X)} \approx \mathrm{pol}(Y)$.

vi) *If* $|V \cap \mathrm{vars}(X)| \approx 0$, *then* $I \models X$ *if and only if* $I/V \models X$.

vii) *If* $|V \cap \mathrm{vars}(X)| \approx 1$, *then* $I \models X$ *if and only if* $I/V \not\models X$.

viii) *The only unsatisfiable parity constraint is* $\mathsf{par}(\top)$.

ix) *The only valid parity constraint is* $\mathsf{par}(\emptyset)$.

x) $I \models X$ *if and only if* $I \not\models X \oplus \mathsf{par}(\top)$.

xi) *If* $A \subseteq B$, *then* $B \models A$.

xii) *If* $A \models B$ *and* $A \models B'$, *then* $A \models B \cup B'$.

xiii) $\mathrm{xor}(l) \equiv l$.

xiv) $A \models A^{\oplus}$

Proof.

i) Observe that $\mathrm{pol}(X \oplus Y) \approx \mathrm{pol}(X) + \mathrm{pol}(Y)$ and:

$$\sum_{a \in \mathrm{vars}(X \oplus Y)} I(a) = \sum_{a \in \mathrm{vars}(X) \triangle \mathrm{vars}(Y)} I(a) \approx \sum_{a \in \mathrm{vars}(X) \triangle \mathrm{vars}(Y)} I(a) + 2 \sum_{a \in \mathrm{vars}(X) \cap \mathrm{vars}(Y)} I(a) = \sum_{a \in \mathrm{vars}(X)} I(a) + \sum_{a \in \mathrm{vars}(Y)} I(a)$$

This means that I satisfies $X \oplus Y$ if and only if:

$$\sum_{a \in \mathrm{vars}(X)} I(a) + \sum_{a \in \mathrm{vars}(Y)} I(a) \approx \mathrm{pol}(X) + \mathrm{pol}(Y) \tag{2.1}$$

Consider in addition the following congruences:

$$\sum_{a\in\mathrm{vars}(X)} I(a) \;\approx\; \mathrm{pol}(X) \qquad (2.2) \qquad\qquad \sum_{a\in\mathrm{vars}(Y)} I(a) \;\approx\; \mathrm{pol}(Y) \qquad (2.3)$$

Now, by Proposition 2.1, we conclude that (2.1) holds if and only if both (2.2) and (2.3) hold or none does. Observing that (2.2) is equivalent to $I \models X$ and (2.3) is equivalent to $I \models Y$ proves the claim.

ii, iii, iv) Follow straightforward from the following identities for arbitrary sets S_1, S_2, S_3:

$$S_1 \triangle S_2 = S_2 \triangle S_1 \qquad S_1 \triangle (S_2 \triangle S_3) = (S_1 \triangle S_2) \triangle S_3 \qquad S_1 \triangle \emptyset = S_1$$

v) A consequence of $\mathrm{vars}(X_{(Y)}) = \mathrm{vars}(X)$ and $\mathrm{pol}(X_{(Y)}) = \mathrm{pol}(Y)$.

vi, vii) Let $J = I/V$, and consider the following congruence:

$$\sum_{a\in\mathrm{vars}(X)} J(a) \;=\; \sum_{a\in\mathrm{vars}(X)\setminus V} I(a) \;+\; \sum_{a\in\mathrm{vars}(X)\cap V} (1 - I(a)) \;\approx\; \sum_{a\in\mathrm{vars}(X)} I(a) + |\mathrm{vars}(X) \cap V|$$

The two claims follow straightforwardly.

viii, ix) $\mathsf{par}(\top)$ is unsatisfiable because

$$\mathrm{pol}(\mathsf{par}(\top)) = 1 \not\approx 0 = \sum_{a\in\emptyset} I(a) = \sum_{a\in\mathrm{vars}(\mathsf{par}(\top))} I(a)$$

and similarly we can check that $\mathsf{par}(\emptyset)$ is valid. Furthermore, if X is a parity constraint containing a variable $a \in \mathrm{vars}(X)$ and I an arbitrary interpretation, then from claims *(vi)* and *(vii)* follows that $I \models X$ if and only if $I/\{a\} \not\models X$, and thus X is neither valid nor unsatisfiable.

x) Straightforward from $\mathrm{pol}(X) \not\approx \mathrm{pol}(X \oplus \mathsf{par}(\top))$.

xi, xii) Straightforward from Definition 2.16.

xiii) If l is a positive literal a, then we obtain $\sum_{b\in\mathrm{vars}(\mathrm{xor}(l))} I(b) = I(a)$ and $\mathrm{pol}(\mathrm{xor}(l)) = 1$, and these two quantities are congruent modulo 2 if and only if $I(a) = 1$, or equivalently $I \models l$. The case when l is a negative literal is similar.

xiv) A consequence of claim *(i)*. □

Claims *(ii)*, *(iii)* and *(ii)* ensure that $\oplus$ is a commutative and associative binary operation over **Par** with $\mathsf{par}(\emptyset)$ as identity, which justifies the following notation. If $A = \{X_1, \ldots, X_n\}$ is an affine formula, then we use the following symbols, similarly to how $\sum$ is used for integer addition:

$$\bigoplus_{i=0}^{0} X_i = \mathsf{par}(\emptyset) \qquad \bigoplus_{i=0}^{j+1} X_i = X_{j+1} \oplus \bigoplus_{i=0}^{j} X_i \qquad \bigoplus_{X\in A} X = \bigoplus_{i=0}^{n}$$

An useful consequence of this notation is the following result, that allows to operate on equalities of parity constraint additions:

Proposition 2.22 (Addition of arbitrary additions):
Let A and B be two affine formulae. Then:

$$\bigoplus_{X\in A} X \oplus \bigoplus_{Y\in B} Y = \bigoplus_{Z\in A\triangle B} Z \qquad (2.4)$$

Proof. Let us enumerate:

$$A \setminus B = \{X_1, \ldots, X_p\} \qquad B \setminus A = \{Y_1, \ldots, Y_q\} \qquad A \cap B = \{Z_1, \ldots, Z_n\}$$

Together with Proposition 2.21, this allows to rewrite (2.4) as:

$$\bigoplus_{i=1}^{p} X_i \oplus \bigoplus_{i=1}^{q} Y_i \oplus \bigoplus_{i=1}^{n} (Z_i \oplus Z_i) = \bigoplus_{i=1}^{p} X_i \oplus \bigoplus_{i=1}^{q} Y_i$$

If we show this equality, then the claim will be proved. Since $\mathrm{vars}(Z_i \oplus Z_i) = \mathrm{vars}(Z_i) \triangle \mathrm{vars}(Z_i) = \emptyset$ and $\mathrm{pol}(Z_i \oplus Z_i) \approx \mathrm{pol}(Z_i) + \mathrm{pol}(Z_i) = 2\mathrm{pol}(Z_i) \approx 0$, we obtain $Z_i \oplus Z_i = \mathsf{par}(\emptyset)$. Again by Proposition 2.21 this implies

$$\bigoplus_{i=1}^{n} (Z_i \oplus Z_i) = \bigoplus_{i=1}^{n} \mathsf{par}(\emptyset) = \mathsf{par}(\emptyset)$$

and therefore

$$\bigoplus_{i=1}^{p} X_i \oplus \bigoplus_{i=1}^{q} Y_i \oplus \bigoplus_{i=1}^{n} (Z_i \oplus Z_i) = \bigoplus_{i=1}^{p} X_i \oplus \bigoplus_{i=1}^{q} Y_i \oplus \mathsf{par}(\emptyset) = \bigoplus_{i=1}^{p} X_i \oplus \bigoplus_{i=1}^{q} Y_i$$

as we wanted. □

Example 2.23 (Semantics of affine formulae):
Consider the following affine formulae, where the parity constraints X_i are given as in Example 2.12:

$$A = \{X_1, X_2, X_3, X_4\} \qquad B = \{X_1, X_2, X_3\}$$

The following hold:

- $A \models B$ straightforward from $B \subseteq A$.
- $B \models \mathsf{par}(\mathsf{p}_3)$ to show this, we show that $\mathsf{par}(\mathsf{p}_3) = X_1 \oplus X_2 \oplus X_3$; entailment then follows from Proposition 2.21:

$$X_1 \oplus X_2 = \mathsf{par}(\mathsf{p}_1, \mathsf{p}_3, \mathsf{p}_4)$$
$$X_1 \oplus X_2 \oplus X_3 = \mathsf{par}(\mathsf{p}_3)$$

- *A is unsatisfiable* assume there is an interpretation I satisfying A. Then, I satisfies B, and by the observation above, I satisfies $\mathsf{par}(\mathsf{p}_3)$ as well. Thus, I must satisfy the parity constraint $\mathsf{par}(\mathsf{p}_3) \oplus X_4$, but this is impossible because $\mathsf{par}(\mathsf{p}_3) \oplus X_4 = \mathsf{par}(\top)$, which is unsatisfiable. Thus, A is unsatisfiable.
- *B is satisfiable* let I be an interpretation with $I(\mathsf{p}_1) = I(\mathsf{p}_3) = I(\mathsf{p}_4) = 0$ and $I(\mathsf{p}_2) = 1$. The parity constraint $X_1 = \mathsf{par}(\mathsf{p}_1, \mathsf{p}_2, \mathsf{p}_3, \top)$ contains an even number of elements satisfied by I, namely p_1 and $\top$, so X_1 is satisfied by I. A similar argument for X_2 and X_3 shows that $I \models B$. Note that, as expected by the observation above, I does not satisfy X_4, which would imply that A is satisfiable.
- $A \not\models B$ follows from the last two observations. ■

The utility of parity constraint addition is given by claim *(i)* in Proposition 2.21. Since parity constraint addition preserves the semantics of affine formulae, proof systems can be constructed using it as an inference rule. In fact, efficient reasoning procedures for affine formula can be built by adapting *Gauss-Jordan elimination* [Day07] to operate over parity constraints. The name of affine formulae is justified by the fact that parity constraints can be regarded as linear equations over the field $\mathbb{Z}_2$ with two elements $0, 1$, where the operations $+, \cdot$ are given by integer addition and product modulo 2. The linear equation corresponding to a parity constraint X is the sum of variables in $\mathrm{vars}(X)$ equated to the integer $\mathrm{pol}(X)$. With this correspondence, any affine formula containing n variables can be regarded as a system of non-homogeneous linear equations, defining an affine subspace of the n-dimensional space over the field $\mathbb{Z}_2$. As for

any other field, Gauss-Jordan elimination-style procedures can be executed over any system of linear equations. Multiplying an equation by 0 yields the trivial equation $0 = 0$, and multiplying it by 1 results in itself. Thus, Gauss-Jordan elimination over $\mathbb{Z}_2$ boils down to addition of equations, which corresponds to $\oplus$-addition for parity constraints. In Section 5.2, a translation of the Gauss-Jordan elimination algorithm to the language of parity constraints is presented.

Example 2.24 (Parity constraints as linear equations):
Consider the affine formula B given in Example 2.23. The correspondence between linear equations and parity constraints in B is shown below:

$$\begin{aligned} \mathsf{p}_1 + \mathsf{p}_2 + \mathsf{p}_3 &\approx 1 & X_1 &= \mathsf{par}(\mathsf{p}_1, \mathsf{p}_2, \mathsf{p}_3, \top) \\ \mathsf{p}_2 + \mathsf{p}_4 &\approx 1 & X_2 &= \mathsf{par}(\mathsf{p}_2, \mathsf{p}_4, \top) \\ \mathsf{p}_1 + \mathsf{p}_4 &\approx 0 & X_3 &= \mathsf{par}(\mathsf{p}_1, \mathsf{p}_4) \end{aligned}$$

Note that an interpretation satisfies each congruence in the left if and only if it satisfies the parity constraint in the right. Now we can perform Gauss-Jordan elimination over the linear system above. Adding equations corresponding to X_1 and X_3 results in the equation corresponding to $X_1 \oplus X_3$:

$$\mathsf{p}_2 + \mathsf{p}_3 + \mathsf{p}_4 \approx 1 \qquad X_1 \oplus X_3 = \mathsf{par}(\mathsf{p}_2, \mathsf{p}_3, \mathsf{p}_4, \top)$$

By adding equations corresponding to X_2 and $X_1 \oplus X_3$ we obtain the equation corresponding to the parity constraint $X_1 \oplus X_2 \oplus X_3$:

$$\mathsf{p}_3 \approx 0 \qquad X_1 \oplus X_2 \oplus X_3 = \mathsf{par}(\mathsf{p}_3)$$

Continuing this process yields a Gauss-Jordan elimination-style procedure. We show below the computation steps.

Input parity constraints:

$$\begin{aligned} X_1 &= \mathsf{par}(\mathsf{p}_1, \mathsf{p}_2, \mathsf{p}_3, \top) & \mathsf{p}_1 + \mathsf{p}_2 + \mathsf{p}_3 &= 1 \\ X_2 &= \mathsf{par}(\mathsf{p}_2, \mathsf{p}_4, \top) & \mathsf{p}_2 + \mathsf{p}_4 &= 1 \\ X_3 &= \mathsf{par}(\mathsf{p}_1, \mathsf{p}_4) & \mathsf{p}_1 + \mathsf{p}_4 &= 0 \end{aligned}$$

Eliminating p_1 from all parity constraints except $\mathsf{par}(\mathsf{p}_1, \mathsf{p}_2, \mathsf{p}_3, \top)$:

$$\begin{aligned} X_1 &= \mathsf{par}(\mathsf{p}_1, \mathsf{p}_2, \mathsf{p}_3, \top) & \mathsf{p}_1 + \mathsf{p}_2 + \mathsf{p}_3 &= 1 \\ X_2 &= \mathsf{par}(\mathsf{p}_2, \mathsf{p}_4, \top) & \mathsf{p}_2 + \mathsf{p}_4 &= 1 \\ X_1 \oplus X_3 &= \mathsf{par}(\mathsf{p}_2, \mathsf{p}_3, \mathsf{p}_4, \top) & \mathsf{p}_2 + \mathsf{p}_3 + \mathsf{p}_4 &= 1 \end{aligned}$$

Eliminating p_2 from all parity constraints except $\mathsf{par}(\mathsf{p}_2, \mathsf{p}_4, \top)$:

$$\begin{aligned} X_1 \oplus X_2 &= \mathsf{par}(\mathsf{p}_1, \mathsf{p}_3, \mathsf{p}_4) & \mathsf{p}_1 + \mathsf{p}_3 + \mathsf{p}_4 &= 0 \\ X_2 &= \mathsf{par}(\mathsf{p}_2, \mathsf{p}_4, \top) & \mathsf{p}_2 + \mathsf{p}_4 &= 1 \\ X_1 \oplus X_2 \oplus X_3 &= \mathsf{par}(\mathsf{p}_3) & \mathsf{p}_3 &= 0 \end{aligned}$$

Eliminating p_3 from all parity constraints except $\mathsf{par}(\mathsf{p}_3)$:

$$\begin{aligned} X_3 &= \mathsf{par}(\mathsf{p}_1, \mathsf{p}_4) & \mathsf{p}_1 + \quad & \quad \mathsf{p}_4 = 0 \\ X_2 &= \mathsf{par}(\mathsf{p}_2, \mathsf{p}_4, \top) & \mathsf{p}_2 + \quad & \quad \mathsf{p}_4 = 1 \\ X_1 \oplus X_2 \oplus X_3 &= \mathsf{par}(\mathsf{p}_3) & \mathsf{p}_3 \quad & \quad = 0 \end{aligned}$$

Observe that the obtained system of linear equations is in reduced row echelon form. In Section 5.2, a correspondent form for affine formulae is introduced. ■

Chapter 3

Proof systems

Our strategy towards unsatisfiability proof generation for parity reasoning in SAT solving consists on building *translations* between the underlying proof system to parity reasoning procedures and that underlying CDCL-style SAT solvers. However, current notions of *proof systems* are not well suited for this goal. In particular, the **Rat** proof system, which has become a standard for unsatisfiability proofs in SAT solving, presents properties whose formalization is problematic, mainly due to **Rat** inferences not preserving semantic equivalence. From a CNF formula, **Rat** is able to derive CNF formulae that are not semantic consequences of the former.

There exists nevertheless an underlying consequence notion to **Rat**-derivations: derived formulae are satisfiable whenever the premises are. This justifies the introduction of a novel notion of abstract proof system operating under arbitrary logics, which correctly models **Rat**. Through the use of such a system, proof systems for parity reasoning can be defined in a natural way. In this chapter, the **Xor** proof system underlying most parity reasoning procedures, such as Gauss-Jordan elimination, is introduced. Furthermore, since our framework allows proof systems sound with respect to cosatisfiability, we obtain an analogue of extended resolution for affine formulae, which we refer to as *extended XOR* (**EXor**).

The rest of this chapter is organized as follows. Section 3.1 is devoted to our formal framework upon which concrete proof systems are defined. Section 3.2 introduces the notion of *resolution asymmetric tautology* and the **Rat** proof system for CNF formulae, together with conditions under which the inference criteria hold. In Section 3.3, the **Xor** and **EXor** proof systems for parity reasoning are presented. Section 3.4 provides tools to compose elementary derivations into more complex ones. The unconventional properties of **Rat** and **EXor** proof systems force us to find conditions under which such compositions are applicable, which is usually granted in more standard proof systems such as resolution and the **Xor** proof system. Finally, in Section 3.5 we review the original contributions in this chapter.

3.1 Abstract proof systems

Proof systems are formalisms that encode consequences as a sequence of simpler, predefined *inference* steps. The value of proof systems becomes apparent when such inferences are mechanically verifiable in an efficient way. While showing that a consequence holds is generally hard due to large search spaces, verifying a derivation is considerably easier: it suffices to check that every inference is correct. A careful design of inferences in such a way that they depend exclusively on syntactical features allows the reduction of consequence verification to a sequence of purely syntactical checks.

There have been a number of proposed formalisms to describe the huge variety of available proof systems. The most widespread is the definition of *propositional proof systems* by Stephen Cook and Robert Reckhow [CR79], which is very appropiate for proof complexity issues. However, this definition fails at

capturing structural properties that are of interest to us, and in particular lacks of a notion of inference. *Abstract theorem proving systems* [BGW94, RP15] were designed for saturation-based proving procedures, which strongly rely on soundness of inferences with respect to the entailment relation, but resolution asymmetric tautology inference violates this condition. From a model-theoretical point of view, some generalizations for proof systems have been proposed in the framework of *institutional logic* [GB92, Dia06], but it is not clear how to adapt institutional logic to the cosatisfiability relation.

The formalization presented here is general enough for our purposes, expressing correctly many proof systems, including *Frege calculi* [Bus98], *resolution* [KL99], *extended resolution* [Tse83], the proof system underlying *Gauss-Jordan elimination* [BM00], and the *DRAT format* for unsatisfiability proofs in SAT solving [WHJ13]. Our formalism is built over a extended notion of logic that allows arbitrary constraints and generalized consequence relations.

Definition 3.1 (Logics):
A logic *is a couple* $\langle L, \models_L \rangle$*, where* L *is a nonempty set and* $\models_L$ *is a binary relation over the set* $\mathcal{F}(L)$ *of finite subsets of* L*, which satisfies:*

i) For all formulae $F \in \mathcal{F}(L)$*, we have* $F_1 \models_L F_1$. (reflexivity)

ii) For all formulae $F, F', F'' \in \mathcal{F}(L)$ *such that* $F \models_L F'$ *and* $F' \models_L F''$*, then* $F \models_L F''$. (transitivity)

iii) For all formulae $F, F', G, G' \in \mathcal{F}(L)$ *such that* $F \subseteq G$*, and* $G' \subseteq F'$*, and* $F \models_L F'$ *hold, then* $G \models_L G'$ *holds as well.* (covariance)

Elements of L *are called* constraints*, and elements of* $\mathcal{F}(L)$*, i.e. finite sets of constraints, are called* formulae*. The relation* $\models_L$ *is called the* consequence relation *of the logic* L*.*

Although this definition models a wide variety of logics, in this work we are only interested in four logics. Regarding languages, we can choose between the language of clauses, which yields formulae in clausal normal form, and the language of parity constraints, which defines formulae as affine systems. The useful consequence relations for our work are the standard entailment relation and the cosatisfiability relation.

Definition 3.2 (Standard/cosatisfiability CNF/XOR logics):
We define the following logics:

- Standard CNF logic*:* $\langle \mathbf{Cla}, \models \rangle$
- Cosatisfiability CNF logic*:* $\langle \mathbf{Cla}, \models_{\mathsf{sat}} \rangle$
- Standard XOR logic*:* $\langle \mathbf{Par}, \models \rangle$
- Cosatisfiability XOR logic*:* $\langle \mathbf{Par}, \models_{\mathsf{sat}} \rangle$

Note that in the former cases we have $\mathcal{F}(\mathbf{Cla}) = \mathbf{Cnf}$*, and in the latter two we have* $\mathcal{F}(\mathbf{Par}) = \mathbf{Aff}$*.*

Proposition 3.3:
The four pairs in Definition 3.2 are logics.

Proof. We show the claim for the cosatisfiability CNF logic; the rest are shown similarly. Reflexivity is straightforward. To show transitivity, assume that $F \models_{\mathsf{sat}} F' \models_{\mathsf{sat}} F''$. If F is satisfiable, then so is F' because $F \models_{\mathsf{sat}} F'$, and similarly F'' is satisfiable because $F' \models_{\mathsf{sat}} F''$, so transitivity follows. Finally, to show covariance, let $F \subseteq G$ and $G' \subseteq F'$, and assume that $F \models_{\mathsf{sat}} F'$. Then, if G is satisfiable, so is any subset of G, in particular F. Satisfiability of F' follows, and since G' is a subset of F', then F' is satisfiable. □

Proof systems are defined over logics. A proof system provides an inference relation that defines admissible derivation steps. Since our system must be general enough to model the DRAT proof format, the

inference relation must be defined over formulae rather than on constraints alone. We only require the inference relation to be sound with respect to the consequence relation in the underlying logic, i.e. inference must yield consequences; and reflexive, which is needed to guarantee that proofs can be concatenated.

Definition 3.4 (Proof systems):
A proof system *is a triple* $R = \langle L, \models_L, \Rightarrow_R \rangle$ *with the following properties:*

i) $\langle L, \models_L \rangle$ *is a logic.*

ii) $\Rightarrow_R$ *is a binary relation over* $\mathcal{F}(L)$.

iii) For all $F \in \mathcal{F}(L)$*, we have* $F \Rightarrow_R F$. (reflexivity)

iv) For all $F, G \in \mathcal{F}(L)$ *such that* $F \Rightarrow_R G$*, we have* $F \models_L G$. (soundness)

The binary relation $\Rightarrow_R$ *is called the* inference relation *of* R.

We define *derivations* as chains of inferences. Transitivity of the consequence relation then provides a way to conclude soundness of derivations, as shown in Proposition 3.7.

Definition 3.5 (Derivations and derivability):
Let $R = \langle L, \models_L, \Rightarrow_R \rangle$ *be a proof system and consider two formulae* F, G*. An* R-derivation *of* G *from* F *is a non-empty string* $\alpha = [H_0, \ldots, H_n]$ *with the following properties:*

i) $F = H_0$.

ii) For all $1 \leq i \leq n$*, we have* $H_{i-1} \Rightarrow_R H_i$.

iii) $H_n = G$.

We denote that α *is an* R*-derivation of* G *from* F *by* $F \mid\!\!\frac{\alpha}{R}\, G$*. In this case,* F *is called the* premises *of* α*, and* G *is called its* conclusions*. We say that* G *is* R-derivable *from* F*, symbolically* $F \mid\!\!\frac{}{R}\, G$*, whenever there exists an* R*-derivation of* G *from* F.

Proposition 3.6 (Concatenation of derivations):
Let $R = \langle L, \models_L, \Rightarrow_R \rangle$ *be a proof system. Let* F, G, H *be three formulae and consider derivations* $F \mid\!\!\frac{\alpha}{R}\, G$ *and* $G \mid\!\!\frac{\beta}{R}\, H$*. Then,* $\alpha : \beta$ *is an* R*-derivation of* H *from* F.

Proof. Let $\alpha = [F_0, \ldots, F_n]$ and $\beta = [F_{n+1}, \ldots, F_m]$ with $m > n$. We show that the conditions in Definition 3.5 hold. Since α is an R-derivation of G from F, then $F_0 = F$; and since β is an R-derivation of H from G, then $F_m = H$. This proves *(i)* and *(iii)*. We now show *(ii)*. Let $1 \leq i \leq m$ be arbitrary. We distinguish three cases:

- If $i \leq n$, then since α is an R-derivation we have $F_{i-1} \Rightarrow_R F_i$ by condition *(ii)*.
- If $i = n$, then observe that $F_n = G$, for α is an R-derivation, and $F_{n+1} = G$, for β is an R derivation; and by reflexivity we obtain $F_n \Rightarrow_R F_{n+1}$.
- If $i > n + 1$, then similarly we obtain $F_{i-1} \Rightarrow_R F_i$ from β being an R-derivation.

Therefore, $\alpha : \beta$ is an R-derivation of H from F. □

Proposition 3.7 (Soundness of $\vdash$):
Let $R = \langle L, \models_L, \Rightarrow_R \rangle$ *be a proof system. Then,* $\mid\!\!\frac{}{R}$ *is the reflexive transitive closure of* $\Rightarrow_R$*. Furthermore, whenever* $F \mid\!\!\frac{}{R}\, G$*, then* $F \models_L G$.

Proof. The first claim follows straigthforwardly from considering that a derivation is just a (possibly trivial) $\Rightarrow_R$-chain. To show that $\mid\!\!\frac{}{R}$ is contained in $\models_L$, recall that $\Rightarrow_R$ is contained in $\models_L$. Thus, the inclusion holds for their respective reflexive transitive closures. The reflexive transitive closure of $\Rightarrow_R$ is $\mid\!\!\frac{}{R}$; the reflexive transitive closure of $\models_L$ is itself. □

Example 3.8 (Consequence proof system):
The *consequence proof system* over a logic $\langle L, \models_L \rangle$ is defined as the triple $\mathbf{Cq}[L] = \langle L, \models_L, \models_L \rangle$. Since $\models_L$ is reflexive, covariant and a subrelation of itself, $\mathbf{Cq}[L]$ is indeed a proof system. Observe that, because $\models_L$ is transitive, $\vdash_{\mathbf{Cq}[L]}$ is exactly $\models_L$. In fact, for every pair of formulae F, G such that $F \models_L G$, a $\mathbf{Cq}[L]$-proof of G from F is given by $[F, G]$. Such proof systems where derivability and entailment relations coincide are called *complete*. ■

The example above shows an extreme case of a proof system that is absolutely useless for theorem proving, or even proof checking. We are interested in proof systems where correct inferences are tractably decidable. Generally, this is attained by providing simple, syntactical operations on formulae whose result is a consequence of the original formula. The most widely known of such operations for CNF formulae is *resolution*.

Example 3.9 (Resolution proof system):
The *resolution proof system* is defined as the triple $\mathbf{Res} = \langle \mathbf{Cla}, \models, \Rightarrow_{\mathbf{Res}} \rangle$, where $F \Rightarrow_{\mathbf{Res}} G$ if some of the following holds:

a) $G \subseteq F$.

b) $G = F \cup F'$, where every clause occurring in F' is a resolvent in F.

The inference relation $\Rightarrow_{\mathbf{Res}}$ is sound with respect to the $\langle \mathbf{Cla}, \models \rangle$ logic due to Proposition 2.19, and as a consequence **Res** is a proof system. However, **Res** is not a complete proof system, i.e. there are CNF formulae F, G such that $F \models G$ but no **Res**-derivation of G from F exists. For example, this is the case if $F = \{\mathsf{or}(\mathsf{p}_1)\}$ and $G = \{\mathsf{or}(\mathsf{p}_1, \mathsf{p}_2)\}$, because there exists no resolvent in F. Nevertheless, the following classical result in propositional logic, found in [KL99], shows that **Res** is very close to be complete:

> If F and G are CNF formulae and $F \models G$, then there is a CNF formula G' such that every clause in G is subsumed by a clause in G', and furthermore G' is **Res**-derivable from F .[1]

Therefore, **Res** can be extended to a complete proof system $\mathbf{IRes} = \langle \mathbf{Cla}, \models, \Rightarrow_{\mathbf{IRes}} \rangle$ where $F \Rightarrow_{\mathbf{IRes}} G$ if some of the following holds:

a') $G \subseteq F$.

b') $G = F \cup F'$, where every clause occurring in F' is a resolvent in F.

c') $G = F \cup F'$, where every clause occurring in F' is subsumed by some clause in F. ■

In the following sections, we introduce the four proof systems that are relevant to our work. The first one is the *resolution asymmetric tautology proof system*, which is widely used in SAT solving to generate unsatisfiability proofs. It is a cosatisfiability-sound extension of the entailment-sound *asymmetric tautology proof system*. The *XOR proof system* for affine formulae, and its *extended XOR proof system* generalization allowing the use of *XOR definitions* are introduced later. The latter two proof systems allow a formalization of parity reasoning as a proving procedure with an underlying proof system.

3.2 Proof systems for CNF formulae

State-of-the-art proof systems for SAT solvers use *redundancy notions* as inference criteria [JHB12]. In general, such redundancy notions satisfy as a basic requirement that clauses learned by SAT solvers must

[1] The result in [KL99, Lemma 4.1.8] is stronger: it shows that every prime implicant of an entailed clause can be derived by resolution. We have not been able to find the original work where this result was first shown.

be redundant in the CNF formula from which they are learned. In general, learned clauses are linear resolvents in the original CNF formula [BKS04]. Modern SAT solvers use in addition many *simplification techniques* [JHB12, MP14, HJB10], and increasingly expressive yet tractable redundancy notions have been studied in order to allow derivations of clauses introduced by such techniques. In particular, *blocked clauses* and *asymmetric tautologies* have been proposed as redundancy notions that allow the construction of proof systems. The current state-of-the-art redundancy notion is that of *resolution asymmetric tautologies*. For a survey on redundancy notions we refer to [JHB12].

Definition 3.10 (Asymmetric literal addition and asymmetric tautologies):
Let F be a CNF formula and L a set of literals. The asymmetric literal addition *function* $\mathrm{ala}_F \colon \mathcal{P}(\mathbf{Lit}) \longrightarrow \mathcal{P}(\mathbf{Lit})$ *is given by:*

$$\mathrm{ala}_F(L) = L \cup \{l \in \mathbf{Lit} : \text{there is a clause } C \in F \text{ such that } \bar{l} \in \mathrm{lits}(C) \text{ and } \mathrm{lits}(C) \setminus \{\bar{l}\} \subseteq L\}$$

Note that ala_F *is an increasing function with respect to the set inclusion. This allows the definition of the limit* $\mathrm{ala}_F^{\infty}(L) = \bigcup_{n \in \mathbb{N}} \mathrm{ala}_F^{n)}(L)$.

A clause C is said to be an asymmetric tautology *in F if* $\mathrm{ala}_F^{\infty}(\mathrm{lits}(C))$ *is inconsistent or F contains the unsatisfiable clause* $\mathsf{or}(\emptyset)$. *Observe that the condition that* $\mathrm{ala}_F^{\infty}(\mathrm{lits}(C))$ *is inconsistent is equivalent to the existence of an $n \in \mathbb{N}$ such that* $\mathrm{ala}_F^{n)}(\mathrm{lits}(C))$ *is inconsistent.*

The set of all asymmetric tautologies in F is denoted by F^{AT}.

Example 3.11 (Asymmetric literal addition):
Consider the CNF formula $F = \{\mathsf{or}(\neg p_1, p_2)\ ,\ \mathsf{or}(p_1, \neg p_3, \neg p_4)\ ,\ \mathsf{or}(\neg p_2, p_4)\}$. We obtain the following iterated applications of ala_F to clauses C_1, C_2, C_3:

C	$C_1 = \mathsf{or}(p_1, \neg p_2, p_3)$	$C_2 = \mathsf{or}(\neg p_1, p_2, p_5)$	$C_3 = \mathsf{or}(p_2, \neg p_3)$
$\mathrm{lits}(C) = \mathrm{ala}_F^{0)}(\mathrm{lits}(C))$	$\{p_1, \neg p_2, p_3\}$	$\{\neg p_1, p_2, p_5\}$	$\{p_2, \neg p_3\}$
$\mathrm{ala}_F^{1)}(\mathrm{lits}(C))$	$\{p_1, \neg p_2, p_3, \neg p_4\}$	$\{\neg p_1, p_1, p_2, \neg p_2, p_5\}$	$\{p_1, p_2, \neg p_3\}$
$\mathrm{ala}_F^{2)}(\mathrm{lits}(C))$	$\{p_1, \neg p_2, p_3, \neg p_3, \neg p_4\}$	$\{\neg p_1, p_1, p_2, \neg p_2, \neg p_4, p_5\}$	$\{p_1, p_2, \neg p_3, \neg p_4\}$
$\mathrm{ala}_F^{3)}(\mathrm{lits}(C))$	$\{p_1, \neg p_2, p_3, \neg p_3, \neg p_4\}$	$\{\neg p_1, p_1, p_2, \neg p_2, \neg p_4, p_5\}$	$\{p_1, p_2, \neg p_3, \neg p_4\}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\mathrm{ala}_F^{\infty}(\mathrm{lits}(C))$	$\{p_1, \neg p_2, p_3, \neg p_3, \neg p_4\}$	$\{\neg p_1, p_1, p_2, \neg p_2, \neg p_4, p_5\}$	$\{p_1, p_2, \neg p_3, \neg p_4\}$
$C \in F^{\mathsf{AT}}$		✓	✓

Since $\mathrm{ala}_F^{\infty}(\mathrm{lits}(C_2))$ and $\mathrm{ala}_F^{\infty}(\mathrm{lits}(C_3))$ are inconsistent, C_2 and C_3 are asymmetric tautologies in F. Observe that both C_2 and C_3 are semantic consequences of F. However, $\mathrm{ala}_F^{\infty}(\mathrm{lits}(C_1))$ is not consistent, so C_1 is not an asymmetric tautology in F. ■

Proposition 3.12 (Basic results about asymmetric literal addition):
Let L, M be sets of literals, F, G be CNF formulae and C be a clause. The following hold:

i) $L \subseteq \mathrm{ala}_F(L)$.

ii) If $L \subseteq M$, *then* $\mathrm{ala}_F(L) \subseteq \mathrm{ala}_F(M)$.

iii) If $F \subseteq G$, *then* $\mathrm{ala}_F(L) \subseteq \mathrm{ala}_G(L)$.

iv) If C is an asymmetric tautology in F and $F \subseteq G$, then C is an asymmetric tautology in G.

Proof.

i, ii, iii) Straightforward from Definition 3.10.

iv) Follows from *(iii)* by induction on the number of applications of the ala function. □

Our definition of asymmetric tautologies differs slightly from others appearing in the literature [JHB12, HJB10, MP14]. In particular, we consider that any clause is an asymmetric tautology in a trivially unsatisfiable CNF formula, i.e. containing the unsatisfiable clause $\mathsf{or}(\emptyset)$. Since the main property of asymmetric tautologies is its soundness, this modification is safe; it furthermore allows to obtain some results like Propositions 3.14 and 3.16 that are violated by degenerate cases with the unmodified definition.

Definition 3.13 (Asymmetric tautology proof system):
Consider the binary relation $\Rightarrow_{\mathbf{At}}$ over the set of CNF formulae **Cnf** *defined by $F \Rightarrow_{\mathbf{At}} G$ if and only if one of the following holds:*

a) $G \subseteq F$. (clause deletion)

b) $G = F \cup F'$ *for some CNF formula* $F' \subseteq F^{\mathsf{AT}}$. (asymmetric tautology introduction)

We define the asymmetric tautology proof system *as the triple* $\mathbf{At} = \langle \mathbf{Cla}, \models, \Rightarrow_{\mathbf{At}} \rangle$.

Proposition 3.14 (Soundness of asymmetric tautology introduction):
Let F be a CNF formula and C an asymmetric tautology in F. Then, $F \models C$.

Proof. Let I be a Boolean interpretation I satisfying F, and assume it does not satisfy C. We show that I satisfies no literal in $\mathrm{ala}_F^{n)}(\mathrm{lits}(C))$ for every $n \in \mathbb{N}$ by induction on n. The case $n = 0$ is straightforward, since $\mathrm{ala}_F^{0)}(\mathrm{lits}(C)) = \mathrm{lits}(C)$ and I does not satisfy C.

Assume now that I satisfies no literal in $\mathrm{ala}_F^{n)}(\mathrm{lits}(C))$ for some $n \in \mathbb{N}$. Let l be any literal from $\mathrm{ala}_F^{n+1)}(\mathrm{lits}(C))$. If $l \in \mathrm{ala}_F^{n)}(\mathrm{lits}(C))$, then by induction hypothesis l is not satisfied by I. Otherwise, there must be a clause $D \in F$ where the literal $\bar{l}$ occurs and $\mathrm{lits}(D) \setminus \{\bar{l}\} \subseteq \mathrm{ala}_F^{n)}(\mathrm{lits}(C))$. Now, D is satisfied by I, but we know that no literal in $\mathrm{ala}_F^{n)}(\mathrm{lits}(C))$ is satisfied by I, which then implies $I \models \bar{l}$. Thus, l is not satisfied by I, and this finishes the proof of the induction step.

By induction, we obtain that no literal in $\mathrm{ala}_F^{\infty}(\mathrm{lits}(C))$ is satisfied by I. Now, C is an asymmetric tautology in F. Hence, either F contains the unsatisfiable clause $\mathsf{or}(\emptyset)$, in which we have a contradiction because I satisfies F; or $\mathrm{ala}_F^{\infty}(\mathrm{lits}(C))$ contains a pair $\{l, \bar{l}\}$ of complementary literals, which is again a contradiction because I satisfies neither of $l, \bar{l}$. By reduction to absurd, I satisfies C. □

Corollary 3.15 (Entailment-soundness of At):
At *is a proof system.*

Proof. We need to check that, for CNF formulae F, G such that $F \Rightarrow_{\mathbf{At}} G$, the entailment $F \models G$ holds.

- If $G \subseteq F$, then straightforwardly $F \models G$.
- If $G = F \cup F'$ for some CNF formula $F' \subseteq F^{\mathsf{AT}}$, then by Proposition 3.14 $F \models C$ for every clause $C \in F'$. Therefore, $F \models F'$, and since $\models$ is reflexive, we obtain $F \models F \cup F' = G$. □

The notion of asymmetric tautology is notably cumbersome, but sufficient conditions exist in terms that are more familiar to the field of propositional logic. In particular, linear resolvents in a CNF formula F are asymmetric tautologies in it, and so are clauses subsumed by some clause in F.

Proposition 3.16 (Linear resolvents are asymmetric tautologies):
Let C be a linear resolvent in a CNF formula F. Then, C is an asymmetric tautology in F.

Proof. Let $[C_0, \ldots, C_n]$ be a linear resolution in F with $C = C_n$. Then, $C_0 \in F$ and there are clauses $D_1, \ldots, D_n \in F$ and literals $l_1, \ldots, l_n$ with $C_i = C_{i-1} \otimes_{l_i} D_i$ for all $1 \leq i \leq n$. Note that $[C_0, \ldots, C_i]$ is a linear resolution in F as well for all $0 \leq i \leq n$. We show by induction on i that C_i is an asymmetric tautology in F for all $0 \leq i \leq n$.

In the base case $i = 0$, we to distinguish two cases. If $C_0 = \mathsf{or}(\emptyset) \in F$, then C_0 is immediately an asymmetric tautology in F. Otherwise, we can find a literal $l \in \mathrm{lits}(C_0)$. Since $\mathrm{lits}(C_0) \setminus \{l\} \subseteq \mathrm{lits}(C_0)$, then we can conclude that $\{l, \bar{l}\} \subseteq \mathrm{ala}_F(C_0)$, so C_0 is an asymmetric tautology in F.

Assume now that for some $0 \leq i < n$ the clause C_i is an asymmetric tautology in F. Then, for some $p \in \mathbb{N}$ we have that $\mathrm{ala}_F^{p)}(C_i)$ is inconsistent. We show that this implies that $C_{i+1} = C_i \otimes_{l_{i+1}} D_{i+1}$ is an asymmetric tautology in F. Let us write $L_j = \mathrm{lits}(C_j)$ for all $0 \leq j \leq n$. Observe that from the inclusion $\mathrm{lits}(D_{i+1}) \setminus \{\overline{l_{i+1}}\} \subseteq \mathrm{lits}(C_{i+1})$ we can conclude $l_{i+1} \in \mathrm{ala}_F(\mathrm{lits}(C_{i+1}))$, and therefore $L_i \subseteq \mathrm{ala}_F(L_{i+1})$. A simple induction using Proposition 3.12 shows that,the inclusion $\mathrm{ala}_F^{j)}(L_i) \subseteq \mathrm{ala}_F^{j+1)}(L_{i+1})$ holds for every $j \in \mathbb{N}$. Hence, we have a situation as depicted below:

$$\begin{array}{ccccccccc} & & L_i & \xmapsto{\mathrm{ala}_F} & \mathrm{ala}_F(L_i) & \xmapsto{\mathrm{ala}_F} & \cdots & \xmapsto{\mathrm{ala}_F} & \mathrm{ala}_F^{p)}(L_i) \\ & & \cap\mathrel{|} & & \cap\mathrel{|} & & & & \cap\mathrel{|} \\ L_{i+1} & \xmapsto{\mathrm{ala}_F} & \mathrm{ala}_F(L_{i+1}) & \xmapsto{\mathrm{ala}_F} & \mathrm{ala}_F^{2)}(L_{i+1}) & \xmapsto{\mathrm{ala}_F} & \cdots & \xmapsto{\mathrm{ala}_F} & \mathrm{ala}_F^{p+1)}(L_{i+1}) \end{array}$$

Since $\mathrm{ala}_F^{p)}(L_i)$ is inconsistent, so is $\mathrm{ala}_F^{p+1)}(L_{i+1})$, which shows that C_{i+1} is an asymmetric tautology in F.

By induction, we have shown that C_i is an asymmetric tautology in F for all $0 \leq i \leq n$, and in particular for $C = C_n$. □

Proposition 3.17 (Subsumed clauses are asymmetric tautologies):
Let F be a CNF formula. Let C be a clause in F and D a clause subsumed by C. Then, D is an asymmetric tautology in F.

Proof. We distinguish two cases:

- If C is the unsatisfiable clause $\mathsf{or}(\emptyset)$, then by Definition 3.10 any clause is an asymmetric tautology in F; in particular, D is.
- If there is a literal $l \in \mathrm{lits}(C) \subseteq \mathrm{lits}(D)$, then we obtain $l \in \mathrm{ala}_F(\mathrm{lits}(D))$. Since D is subsumed by C, we know that $\mathrm{lits}(C) \setminus \{l\} \subseteq \mathrm{lits}(D)$. By definition of the ala_F function, this implies that $\bar{l} \in \mathrm{ala}_F(\mathrm{lits}(D))$. Therefore, $\mathrm{ala}_F(\mathrm{lits}(D))$ is inconsistent, and D is an asymmetric tautology in F. □

Asymmetric tautologies have been regarded as a suitable redundancy notion to express resolution derivations, since linear resolvents can be compressed in a single step. As will be shown in Section 4.1.3, learned clauses in CDCL-style SAT solving are asymmetric tautologies as well, which allows to express unsatisfiability proofs in classical CDCL-style SAT solving (i.e. without inprocessing rules and using only clauses in the current CNF formula as reason clauses during conflict analysis) merely as a sequence of learned clauses.

3.2.1 Resolution Asymmetric Tautologies

The proof systems presented so far are sound for the standard entailment relation over **Cnf**, and hence for the weaker cosatisfiability relation. However, there exist other proof systems that are sound for the latter but not for the former. Although we do not present it here, the *extended resolution proof* system [Tse83] can be modeled in our proof system framework. In extended resolution, fresh literals (i.e. literals whose underlying variable is used nowhere else in the derivation) can be used to introduce definitions. This allows an exponential reduction in the maximum size of derivations [CR79], and so extended resolution is regarded as one of the most powerful proof systems from the proof complexity perspective. We present here a generalization of the extended resolution proof system, called *resolution asymmetric tautology*.

Due to soundness with respect to the entailment relation, asymmetric tautologies are uncapable of expressing inprocessing techniques that may not preserve semantic equivalence but rather equisatisfiability. For example, introducing definitions as in the extended resolution proof system is only cosatisfiability-

sound, and thus not covered by asymmetric tautologies. The notion of *resolution asymmetric tautologies* was introduced to allow the introduction of clauses in a satisfiability-preserving way.

Definition 3.18 (Resolution asymmetric tautology proof system):
A clause C is said to be a resolution asymmetric tautology *in a CNF formula F upon a literal $l \in \mathrm{lits}(C)$ if, for every clause $D \in F$ such that C is resolvable with D upon l, the resolvent $C \otimes_l D$ is an asymmetric tautology in F. We say that C is a resolution asymmetric tautology in F if there is a literal l satisfying this condition. We denote the set of resolution asymmetric tautologies in F by F^{RAT}.*

Consider the binary relation $\Rightarrow_{\mathbf{Rat}}$ over the set of CNF formulae $\mathbf{Cnf}$ defined by $F \Rightarrow_{\mathbf{Rat}} G$ if and only if one of the following holds:

a) $G \subseteq F$. (clause deletion)

b) $G = F \cup F'$ *for some CNF formula* $F' \subseteq F^{\mathsf{AT}}$. (asymmetric tautology introduction)

c) $G = F \cup F'$ *for some unresolvable CNF formula* $F' \subseteq F^{\mathsf{RAT}}$. (resolution asymmetric tautology introduction)

We define the resolution asymmetric tautology proof system *as the triple* $\mathbf{Rat} = \langle \mathbf{Cla}, \models_{\mathsf{sat}}, \Rightarrow_{\mathbf{Rat}} \rangle$.

The **Rat** proof system corresponds to the state-of-the-art proof format, called *Deletion Resolution Asymmetric Tautologies (DRAT)*. **Rat** proofs are so expressive that they can polynomially simulate extended resolution proofs [MP14]. There is a subtle difference between the usual definition for DRAT and our **Rat** proof system, namely that in the DRAT standard clauses are added or removed one by one, whereas in the **Rat** we allow the addition of whole CNF formulae in ever inference step. The relation between the DRAT proof format and **Rat**-derivations is discussed in Section 4.2.1.

To show that **Rat** is a proof system, we show a more general result that allows the construction of cosatisfiability-sound proof systems from arbitrary entailment-sound proof systems for CNF formulae.

Theorem 3.19 (Extension by resolution of entailment-sound proof systems):
Let $\Rightarrow_R$ be a binary relation on $\mathbf{Cnf}$ such that $R = \langle \mathbf{Cla}, \models, \Rightarrow_R \rangle$ is a proof system, and define a binary relation $\Rightarrow_{R'}$ on $\mathbf{Cnf}$ such that $F \Rightarrow_{R'} G$ if and only if one of the following holds:

a) $F \Rightarrow_R G$.

b) $G = F \cup F'$ *for some unresolvable CNF formula F' such that, for every clause $C \in F'$ there exists a literal $l \in \mathrm{lits}(C)$ with the following condition:*

$$\text{For every } D \in F \text{ such that } C \text{ is resolvable with } D \text{ upon } l, \text{ we have } F \Rightarrow_R F \cup \{C \otimes_l D\}. \tag{3.1}$$

Then, $R' = \langle \mathbf{Cla}, \models_{\mathsf{sat}}, \Rightarrow_{R'} \rangle$ is a proof system.

Proof. Since $\Rightarrow_R$ is reflexive, reflexivity of $\Rightarrow_{R'}$ follows. We now show soundness. Let F and G be two CNF formulae such that $F \Rightarrow_{R'} G$. We show that $F \models_{\mathsf{sat}} G$ holds for each of the cases in the claim.

- If $G \Rightarrow_R F$, then since R is a proof system sound with respect to entailment, $F \models G$, and consequently $F \models_{\mathsf{sat}} G$.
- If condition *(b)* holds, then let us enumerate F' as $\{C_1, \dots, C_n\}$ and, for every $0 \le i \le n$, consider the CNF formula $G_i = F \cup \{C_1, \dots, C_i\}$. We show that $F \models_{\mathsf{sat}} G_i$ for all $0 \le i \le n$ by induction in i. The case $i = 0$ is immediate, for $F = G_0$. We now assume that $F \models_{\mathsf{sat}} G_i$ for some $0 \le i < n$, and show $F \models_{\mathsf{sat}} G_{i+1}$.

  Assume that F is satisfiable. Then, so is G_i, thus we can find an interpretation I satisfying G_i. We know that there is a literal $l \in \mathrm{lits}(C_{i+1})$ such that C_{i+1} satisfies condition (3.1). Let $a = \mathrm{var}(l)$; we

show that either I or $J = I/\{a\}$ satisfies G_{i+1} by reduction to absurd. Assume that none of them does. Since I satisfies $G_i = G_{i+1} \setminus \{C_{i+1}\}$, we conclude that $I \not\models C_{i+1}$. In particular, $I \not\models l$, so we obtain $J \models l$ and consequently $J \models C_{i+1}$. Since J does not satisfy G_{i+1}, there must be a clause $D \in G_i$ which is falsified by J. Let us show that C is resolvable with D upon l.

- $l \in \mathrm{lits}(C_{i+1})$ *holds* already shown.
- $\bar{l} \in \mathrm{lits}(D)$ *holds* since J falsifies D, then J falsifies every literal in D. However, I satisfies D because I satisfies G_i. This can only be the case if $\bar{l} \in \mathrm{lits}(D)$.
- *For every literal* $k \in \mathrm{lits}(C_{i+1}) \setminus \{l\}$*, its complement* $\bar{k}$ *does not occur in* D if such a literal existed, then since I falsifies C_{i+1}, it would satisfy $\bar{k}$ and therefore D. Now, $\mathrm{var}(k) \neq a$, which would imply that J satisfies D as well, but it does not.
- *For every literal* $k \in \mathrm{lits}(D) \setminus \{\bar{l}\}$*, its complement* $\bar{k}$ *does not occur in* C_{i+1} follows similarly to the previous property, because J falsifies D and I falsifies C_{i+1}.

Hence, we obtain that C_{i+1} is resolvable with D upon l. Since F' is unresolvable, we conclude that $C_{i+1} \in F$. Now, by property (3.1), we have $F \Rightarrow_R F \cup \{C_{i+1} \otimes_l D\}$, and since R is a proof system this means that $F \models C_{i+1} \otimes_l D$. Now, I satisfies $G_i \supseteq F$, so I satisfies $C_{i+1} \otimes D$. Let k be a literal satisfied by I that occurs in $C_{i+1} \otimes D$. Observe that k is neither l or $\bar{l}$, so $I(k) = J(k)$. But k occurs either in C_{i+1} or in D, which means that either clause is satisfied by both I and J, which is a contradiction.

Therefore, we conclude that either I or J satisfy G_{i+1}, hence G_{i+1} is satisfiable. By induction, we conclude that $G_n = F \cup F' = G$ is satisfiable, so $F \models\!\!\!\models_{\mathsf{sat}} G$. □

Corollary 3.20 (Cosatisfiability-soundness of Rat):
Rat *is a proof system.*

Proof. Follows from observing that, in Theorem 3.19, if $R = \mathbf{At}$ then $R' = \mathbf{Rat}$. □

3.3 Proof systems for affine formulae

The proof systems presented so far operate over the standard or cosatisfiability CNF logics. Our approach to generate unsatisfiability proofs for parity reasoning is based on translating execution records of Gauss-Jordan elimination procedures into **Rat**-derivations. To do so, we model Gauss-Jordan elimination as a theorem proving procedure with an underlying proof system, whose main inference rule consists on parity constraint addition. The idea of considering parity constraint addition as part of a proof system can be backtracked to Peter Baumgartner and Fabio Massacci [BM00], who presented a proof system with a Gauss resolution inference rule similar to parity constraint addition.

As described in Section 2.2, Gauss-Jordan elimination over affine formulae can be assimilated to the homonymous procedure in linear algebra executed over linear systems with coefficients in $\mathbb{Z}_2$. In this setting, linear combinations of equations can be obtained using exclusively sums (modulo 2) of equations, which correspond to parity constraint addition. This justifies the introduction of the following proof system:

Definition 3.21 (XOR proof system):
Consider the binary relation $\Rightarrow_{\mathbf{Xor}}$ *over the set of affine formulae* **Aff** *defined by* $A \Rightarrow_{\mathbf{Xor}} B$ *if and only if one of the following holds:*

a) $B \subseteq A$. (parity constraint deletion)

b) $B = A \cup \{Y \oplus Z\}$ *for some parity constraints* $Y, Z \in A$. (parity constraint addition)

We define the XOR proof system *as the triple* $\mathbf{Xor} = \langle \mathbf{Par}, \models, \Rightarrow_{\mathbf{Xor}} \rangle$.

Proposition 3.22 (Entailment-soundness of Xor):
Xor *is a proof system.*

Proof. A direct consequence of Proposition 2.21. □

The correspondence between parity reasoning, Gauss-Jordan elimination and the **Xor**-proof system will be shown later in Theorem 5.12. A consequence of this result is that **Xor** can derive the unsatisfiable parity constraint $\mathsf{par}(\top)$ from any unsatisfiable affine formula. However, the **Xor** proof system has some limitations that will become apparent when linear translations of XOR derivations are presented in Chapter 7. **Xor** is only sound for the standard XOR logic, thus excluding cosatisfiability proofs. This hinders its capability to produce derivations that will be of interest to us. For one, given two disjoint sets of pseudovariables V and W and a variable $a \notin V \cup W$, the parity constraint $\mathsf{par}(V \cup W)$ is equisatisfiable to the affine formula $\{\mathsf{par}(V) \oplus \mathsf{par}(a)\ ,\ \mathsf{par}(W) \oplus \mathsf{par}(a, \top)\}$, but the latter affine formula can not be **Xor**-derived from $\{\mathsf{par}(V \cup W)\}$. In Section 5.1 we present encodings of parity constraints as CNF formulae that strongly rely in this property, and to derive such encodings we will need a more suitable proof system.

The *extended XOR proof system* is based on the idea of introducing definitions with Tseitin variables in extended resolution. Within our formal framework for proof systems, generalizing the XOR proof system in the same direction can be naturally done.

Definition 3.23 (Extended XOR proof system):
A parity constraint X is called an XOR definition *in an affine formula A upon a variable a if $a \in \mathrm{vars}(X) \setminus \mathrm{vars}(A)$. We simply say that X is an XOR definition in A if there is a variable a that satisfies this condition.*

Consider the binary relation $\Rightarrow_{\mathbf{EXor}}$ over the set of affine formulae **Aff** *defined by $A \Rightarrow_{\mathbf{EXor}} B$ if and only if one of the following holds:*

a) $B \subseteq A$. (parity constraint deletion)

b) $B = A \cup \{Y \oplus Z\}$ *for some parity constraints* $Y, Z \in A$. (parity constraint addition)

c) $B = A \cup \{X\}$ *for a XOR definition X in A.* (XOR definition introduction)

We define the extended XOR proof system *as the triple* $\mathbf{EXor} = \langle \mathbf{Par}, \models_{\mathsf{sat}}, \Rightarrow_{\mathbf{EXor}} \rangle$.

We now show that **EXor** is indeed a cosatisfiability-sound proof system.

Lemma 3.24 (Cosatisfiability-soundness of XOR definition introduction):
Let A be a satisfiable affine formula and X an XOR definition in A. Then, $A \cup \{X\}$ is satisfiable.

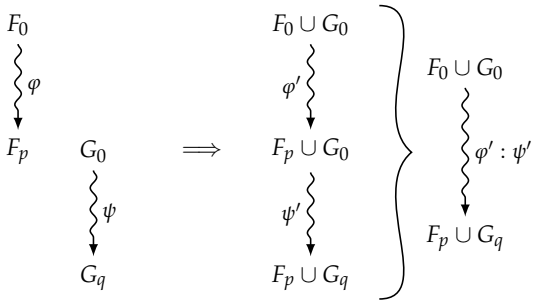
Proof. Let I be an interpretation satisfying A. If I satisfies X, then the conclusion follows. Otherwise, I does not satisfy X. Since X is an XOR definition in A, there is a variable $a \in \mathrm{vars}(X)$ such that $a \notin \mathrm{vars}(A)$. Define the interpretation $J = I/\{a\}$. Since $a \notin \mathrm{vars}(A)$, the interpretation J satisfies A as well. Now, since the number of elements in X satisfied by I and those satisfied by J differ by exactly 1, we conclude that J satisfies X if and only if I does not satisfy X. The right hand side of this equivalence holds, and therefore $A \cup \{X\}$ is satisfied by J. □

Corollary 3.25 (Cosatisfiability-soundness of EXor):
EXor *is a proof system.*

Proof. Let A and B be two affine formulae with $A \Rightarrow_{\mathbf{EXor}} B$. Then, either of the conditions in Definition 3.23 holds. If conditions *(a)* or *(b)* hold, then $A \models B$, therefore $A \models_{\mathsf{sat}} B$. Otherwise, condition *(c)* holds, and then $A \models_{\mathsf{sat}} B$ follows by Lemma 3.24. □

3.4 Derivation composition

One of the main issues when applying our framework to unsatisfiability proof generation that derivations might not be easy to compose with each other. In entailment-sound proof systems (that is, proof systems over logics with the entailment as consequence relation), derivations can be composed to construct larger derivations. For example, one might have a derivation φ of a formula F_p from another formula F_0, and a second derivation ψ of a formula G_q from another formula G_0. For traditional proof systems, such as resolution or **At**, one can always find a derivation of $F_q \cup G_p$ from $F_0 \cup G_0$, as depicted in the following diagram:



This sort of diagrams will be frequently used through this Thesis and offer a very convenient way to depict operations over derivations. "Wave" arrows from a formula F to a formula G represent derivations of G from F. A label in a wave arrow represents a symbol denoting that derivation. Braces engulfing a sequence of derivations are used to represent the concatenation of the enclosed derivations.

Most proof systems that are sound with respect to the entailment relation allow the construction of derivations φ' and ψ' as above. For example, if $\varphi = [F_0, \ldots, F_p]$ and $\psi = [G_0, \ldots, G_q]$ are **At**-derivations, then so are the strings of CNF formulae

$$\varphi' = [F_0 \cup G_0, F_1 \cup G_0, \ldots, F_p \cup G_0] \qquad \psi' = [F_p \cup G_0, F_p \cup G_1, \ldots, F_p \cup G_q]$$

and then $\varphi' : \psi'$ is an **At**-derivation of $F_p \cup G_q$ from $F_0 \cup G_0$. In this Thesis, we call this property *deepness*. The name is motivated by similarity with *deep inference* [Gug99], where inference rules can be applied independently of contexts.[2]

Definition 3.26 (Deep proof systems):
Let $R = \langle L, \models_L, \Rightarrow_R \rangle$ be a proof system. R is said to be deep *if, for every three formulae F, G, H, whenever $F \Rightarrow_R G$ holds, $F \cup H \Rightarrow_R G \cup H$ holds as well.*

Proposition 3.27 (Examples of deep proof systems):
The proof systems **At** *and* **Xor** *are deep.*

Proof. First we show that **At** is deep. Consider two CNF formulae F, G with $F \Rightarrow_{\mathbf{At}} G$, and an additional CNF formula H. One of the following holds:

- If $G \subseteq F$, then clearly $G \cup H \subseteq F \cup H$.
- If $G = F \cup F'$ for a CNF formula $F' \subseteq F^{\mathsf{AT}}$, then every clause $C \in F'$ is an asymmetric tautology in F, and by Proposition 3.12, it is an asymmetric tautology in $F \cup H$ as well. Then, $F' \subseteq (F \cup H)^{\mathsf{AT}}$.

In both cases we conclude that $F \cup H \Rightarrow_{\mathbf{At}} G \cup H$, so **At** is a deep proof system.

Now we show that **Xor** is deep. As above, consider two affine formulae A, A' with $A \Rightarrow_{\mathbf{Xor}} A'$, and an additional affine formula B. One of the following holds:

[2] Deep inference works in a more structural proof theory framework than ours. Our notion of deepness is adapted to the framework presented in this chapter, and is not interchangeable with that from deep inference.

- If $A' \subseteq A$, then $A' \cup B \subseteq A \cup B$.
- If $A' = A \cup \{Y \oplus Z\}$ for some parity constraints $Y, Z \in A \subseteq A \cup B$, then we have the equality $A' \cup B = (A \cup B) \cup \{Y \oplus Z\}$.

In both cases we obtain $A \cup B \Rightarrow_{\mathbf{Xor}} A'B$, so **Xor** is a deep proof system. □

The notion of deepness seems quite a natural requirement for a proof system, but two proof systems we are interested in violate it, namely **Rat** and **EXor**. Since the **At** and **Xor** proof systems are deep, the problematic inferences are resolution asymmetric tautology introduction for the **Rat** proof system, and XOR definition introduction for the **EXor** proof system. Examples 3.28 and 3.29 show that the proof systems **Rat** and **EXor** are not deep.

Example 3.28 (Non-deepness of Rat):
Consider the following clauses:

$$C_1 = \mathsf{or}(\mathsf{p}_1, \mathsf{p}_2) \qquad C_2 = \mathsf{or}(\neg\mathsf{p}_1, \neg\mathsf{p}_2) \qquad C_3 = \mathsf{or}(\neg\mathsf{p}_1, \mathsf{p}_2)$$

The inference $F = \{C_1\} \Rightarrow_{\mathbf{Rat}} \{C_1, C_2\} = F'$ holds by resolution asymmetric tautology introduction: C_2 is a resolution asymmetric tautology in $\{C_1\}$ upon $\neg\mathsf{p}_1$ because C_2 is not resolvable with C_1.

However, the inference $G = F \cup \{C_3\} = \{C_1, C_3\} \Rightarrow_{\mathbf{Rat}} \{C_1, C_2, C_3\} = F' \cup \{C_3\} = G'$ does not hold:

- The inference above does not hold by clause deletion, since G' is not a subset of G.
- It does not hold by asymetric tautology addition either, because C_2 is not an asymmetric tautology in G. This follows from $\mathrm{ala}_G(\mathrm{lits}(C_2)) = \mathrm{ala}_G\{\neg\mathsf{p}_1, \neg\mathsf{p}_2\} = \{\neg\mathsf{p}_1, \neg\mathsf{p}_2\}$. Since $\mathrm{lits}(C_2)$ is a fixpoint for ala_G, we obtain $\mathrm{ala}_G^\infty(\mathrm{lits}(C)) = \mathrm{lits}(C)$, which is a consistent set of literals.
- Neither does it hold by resolution asymmetric tautology addition. C_2 is not resolvable with C_1, since there are two pairs of complementary literals in their literal sets. Furthermore, even though C_2 is resolvable with C_3 upon $\neg\mathsf{p}_2$, the resolvent $D = C_2 \otimes_{\neg\mathsf{p}_2} C_3 = \mathsf{or}(\neg\mathsf{p}_1)$ is not an asymmetric tautology in G, as the following computation of ala_G shows:

$$\begin{aligned} \mathrm{ala}_G^{0)}(\mathrm{lits}(D)) &= \{\neg\mathsf{p}_1\} \\ \mathrm{ala}_G^{1)}(\mathrm{lits}(D)) &= \{\neg\mathsf{p}_1, \neg\mathsf{p}_2\} \\ \mathrm{ala}_G^\infty(\mathrm{lits}(D)) = \mathrm{ala}_G^{2)}(\mathrm{lits}(D)) &= \{\neg\mathsf{p}_1, \neg\mathsf{p}_2\} \end{aligned}$$

■

Example 3.29 (Non-deepness of EXor):
Consider the following parity constraints:

$$X_1 = \mathsf{par}(\mathsf{p}_1) \qquad X_2 = \mathsf{par}(\mathsf{p}_2) \qquad X_3 = \mathsf{par}(\mathsf{p}_1, \mathsf{p}_2, \top)$$

The inference $A = \{X_1\} \Rightarrow_{\mathbf{EXor}} \{X_1, X_2\} = A'$ holds by XOR definition introduction, since $\mathsf{p}_2 \notin \mathrm{vars}(A)$. However, the inference $B = A \cup \{X_3\} = \{X_1, X_3\} \Rightarrow_{\mathbf{EXor}} \{X_1, X_2, X_3\} = A' \cup \{X_3\} = B'$ does not hold:

- The inference above does not hold by parity constraint deletion, since B' is not a subset of B.
- It does not hold by parity constraint addition, since the only parity constraints that can be obtained by addition from B are $\mathsf{par}(\emptyset)$ and $\mathsf{par}(\mathsf{p}_2, \top)$.
- Neither does it hold by XOR definition introduction, because the only variable in X_2, namely p_2, already occurs in B. ■

Our main composition operation is called *fusion*. Intuitively, fusion of derivations yields a new, larger derivation of all conclusions of component derivations from all their premises, provided that the proof

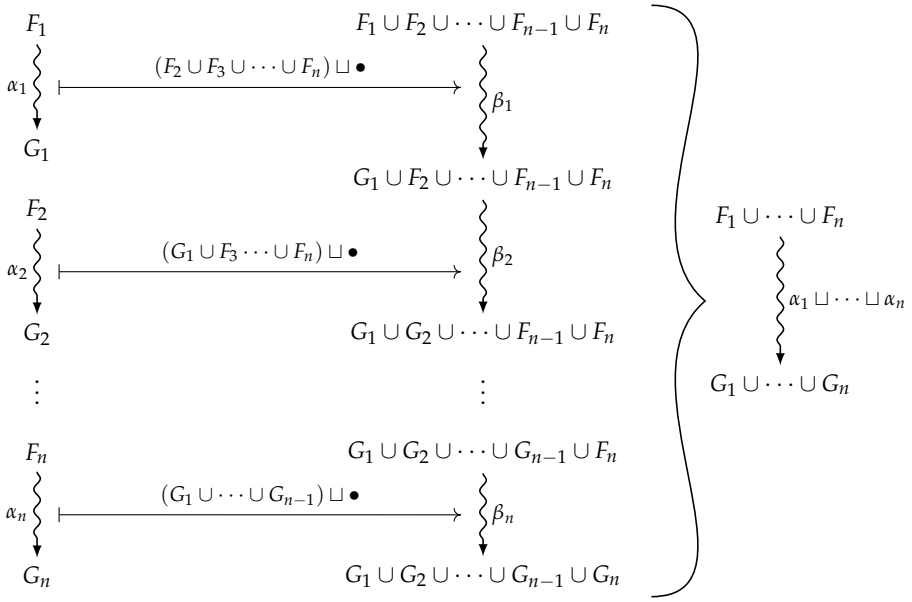


Figure 3.1: Construction of the fusion of derivations $\alpha_1 \sqcup \cdots \sqcup \alpha_n$. Each derivation α_i is first fused with all conclusions of the α_j with $j < i$, and all premises of the α_j with $j > i$ to obtain a derivation β_i. The β_i derivations can be concatenated, because the premises of each β_i are the conclusions of β_{i-1}. The fusion $\alpha_1 \sqcup \cdots \sqcup \alpha_n$ results from this concatenation. "Wave" arrows represent derivations, braces denote concatenation, and $\mapsto$ arrows represent the application of a mapping whose argument is denoted by $\bullet$.

system used is deep. To some extent, fusion of derivations can be regarded as an union of independent derivations.

Definition 3.30 (Fusion of a formula into a derivation):
Let $R = \langle L, \models_L, \Rightarrow_R \rangle$ be a proof system, and $\alpha = [F_0, \ldots, F_n]$ be a string of formulae. For every formula H, we define the fusion *of H into α as the string of formulae $H \sqcup \alpha = [F_0 \cup H, \ldots, F_n \cup H]$.*

Definition 3.31 (Fusion of derivations):
Let $R = \langle L, \models_L, \Rightarrow_R \rangle$ be a proof system, and consider strings of formulae $\alpha_1, \ldots, \alpha_r$ such that α_i is an R-derivation of a formula G_i from a formula F_i for every $1 \leq i \leq r$. Define β_i as the string of formulae given by:

$$\beta_i = \left(\bigcup_{j=0}^{i-1} F_j \cup \bigcup_{j=i+1}^{n} G_j \right) \sqcup \alpha_i$$

The fusion *of $\alpha_1, \ldots, \alpha_r$ is defined as the string of formulae $\alpha_1 \sqcup \cdots \sqcup \alpha_r = \beta_1 : \ldots : \beta_r$.*

The notion of fusion might be difficult to grasp from the formal definition, although the intuition behind is rather straightforward. Fusion is an extended notion of concatenation for derivations that do not coincide in their premises. Figure 3.1 shows how concatenation works. First, every derivation α_i is transformed into β_i by fusing it with the formula $F_1 \cup \ldots F_{i-1} \cup G_{i+1} \cup \ldots G_n$, that is, joining this formula to every formula along α_i. Observe that, after this is done, the resulting strings β_i have matching ends. Starting from the formula $F_1 \cup \cdots \cup F_n$, every β_i derivation replaces one of the F_i by G_i, until β_n derives the formula $G_1 \cup \cdots \cup G_n$. Concatenating these strings together yields the fusion string $\alpha_1 \sqcup \cdots \sqcup \alpha_n$.

In deep proof systems, fusion of derivations straightforwardly yields derivations. As shown later in Corollaries 3.38 and 3.39, in the **Xor** and **At** proof systems derivations can be arbitrarily fused. We are interested in finding sufficient conditions that allow fusion in the **EXor** and **Rat** proof systems, since they

are more general than the former. To keep track of composition of derivations, we introduce the notion of *sharpness*. Intuitively, a derivation α is sharp in a set of variables V whenever fusion with other proofs be problematic only due to variables in V.

Definition 3.32 (Sharpness sets for Rat-derivations):
Let $\alpha = [F_0, \ldots, F_n]$ be a **Rat***-derivation and V a set of variables. α is said to be* sharp *in V if, for all $1 \leq i \leq n$, some of the following holds:*

a) $F_{i-1} \Rightarrow_{\mathbf{At}} F_i$.

b) $F_i = F_{i-1} \cup F'$ *for some unresolvable CNF formula $F' \subseteq F_{i-1}$ such that every clause $C \in F'$ is a resolution asymmetric tautology in F_{i-1} upon a literal l with* $\mathrm{var}(l) \in V$.

Definition 3.33 (Sharpness sets for EXor-derivations):
Let $\varphi = [A_0, \ldots, A_n]$ be an **EXor***-derivation and V a set of variables. φ is said to be* sharp *in V if, for all $1 \leq i \leq n$, some of the following holds:*

a) $A_{i-1} \Rightarrow_{\mathbf{Xor}} A_i$.

b) $A_i = A_{i-1} \cup \{X\}$ *for some parity constraint X such that X is an XOR definition in A_{i-1} upon a variable $a \in V$.*

These notions of sharpness allow us to find useful conditions under which fusion of derivations yields again a derivation for the **Rat** and **EXor** proof systems. We present three results. Proposition 3.34 shows how sharpness behaves upon derivation concatenation. Propositions 3.35 and 3.36 provide conditions for fusion with formulae to be derivations, whereas Proposition 3.37 does so for fusion between derivations.

Proposition 3.34 (Sharpness and concatenation): *i) Let α be a* **Rat***-derivation of a CNF formula F' from a CNF formula F, and β a* **Rat***-derivation of a CNF formula F'' from F'. If α is sharp in V and β is sharp in W, then their concatenation $\alpha : \beta$ is sharp in $V \cup W$.*

ii) Let φ be a **EXor***-derivation of an affine formula A' from a CNF formula A, and ψ a* **Rat***-derivation of a CNF formula A'' from A'. If φ is sharp in V and ψ is sharp in W, then their concatenation $\varphi : \psi$ is sharp in $V \cup W$.*

Proof. Straightforward from Definitions 3.32 and 3.33. □

Proposition 3.35 (Sharpness and fusion of CNF formulae into Rat-derivations):
Let $\alpha = [F_0, \ldots, F_n]$ be a **Rat***-derivation of F_n from F_0 sharp in V, and H a CNF formula. If* $\mathrm{vars}(H)$ *is disjoint from V, then $H \sqcup \alpha$ is a* **Rat***-derivation of $F_n \cup H$ from $F_0 \cup H$ sharp in V.*

Proof. The derivation $H \sqcup \alpha$ is given by $[H \cup F_0, \ldots, H \cup F_n]$. We first show that $H \sqcup \alpha$ is a **Rat**-derivation. For this, it suffices to show that $H \cup F_{i-1} \Rightarrow_{\mathbf{Rat}} H \cup F_i$ for every $1 \leq i \leq n$. By Definition 3.32, we know that some of the following holds:

a) $F_{i-1} \Rightarrow_{\mathbf{At}} F_i$. In this case, by Proposition 3.27, $H \cup F_{i-1} \Rightarrow_{\mathbf{At}} H \cup F_i$ holds.

b) $F_i = F_{i-1} \cup F'$ for some unresolvable CNF formula $F' \subseteq F_{i-1}$ such that every clause $C \in F'$ is a resolution asymmetric tautology in F_{i-1} upon a literal l with $\mathrm{var}(l) \in V$. Let C be an arbitrary clause in F' with that property. We show the following:

$$C \text{ is an asymmetric resolution tautology in } F_{i-1} \cup H \text{ upon } l. \tag{3.2}$$

Let D be a clause in $F_{i-1} \cup H$ such that C is resolvable with D upon l. Assume that $D \in H$. By resolvability, the literal $\bar{l}$ occurs in the clause D, and in particular $\mathrm{var}(l) \in \mathrm{vars}(D) \subseteq \mathrm{vars}(H)$. But this leads to a contradiction, insofar as $\mathrm{vars}(H)$ and V are disjoint. We conclude that $D \notin H$, so $D \in F_{i-1}$ must hold. Then, since C is a resolution asymmetric tautology in F_{i-1} upon l, we conclude

that the resolvent $C \otimes_l D$ is an asymmetric tautology in F_{i-1}. By Proposition 3.12, it is an asymmetric tautology in $F_{i-1} \cup H$ as well. Therefore, (3.2) holds.

Therefore, we have shown that F' is an unresolvable CNF formula with $F' \subseteq (F_{i-1} \cup H)^{\mathsf{RAT}}$, so the inference $H \cup F_{i-1} \Rightarrow_{\mathbf{Rat}} H \cup F_{i-1} \cup F' = H \cup F_{i-1}$ holds.

Hence, $H \sqcup \alpha$ is a **Rat**-derivation of $H \cup F_n$ from $H \cup F_0$. It remains to see that $H \sqcup \alpha$ is sharp in V. This follows from (3.2), since resolution asymmetric tautologies in F_{i-1} upon a literal in V are resolution asymmetric tautologies in $F_{i-1} \cup H$ in that same literal. □

Proposition 3.36 (Sharpness and fusion of affine formulae into EXor-derivations):
Let $\varphi = [A_0, \ldots, A_n]$ be an **EXor***-derivation of A_n from A_0 sharp in V, and B an affine formula. If* $\mathrm{vars}(B)$ *is disjoint from V, then $B \sqcup \varphi$ is an* **EXor***-derivation of $A_n \cup B$ from $A_0 \cup B$ sharp in V.*

Proof. The proof is completely analogous to that of Proposition 3.35 by replacing resolution asymmetric tautologies by XOR definitions and the **At** proof system by **Xor**. The only noticeably different argument is showing that if $A_i = A_{i-1} \cup X$ for some XOR definition X in A upon a variable $a \in V$, then X is an XOR definition as well in $A_{i-1} \cup B$ upon a. This follows straightforward from $\mathrm{vars}(B)$ and V are disjoint, so $a \notin \mathrm{vars}(B)$. □

Proposition 3.37 (Sharpness and fusion of derivations): *i) Let α_i be a* **Rat***-derivation of a CNF formula G_i from another CNF formula F_i sharp in a set of variables V_i for every $1 \leq i \leq n$. Assume that, for every $1 \leq i, j \leq n$ with $i \neq j$, the set of variables* $\mathrm{vars}(F_j \cup G_j)$ *is disjoint from V_i. Then, $\alpha_1 \sqcup \cdots \sqcup \alpha_n$ is a* **Rat***-derivation of $\bigcup_{i=1}^n G_i$ from $\bigcup_{i=1}^n F_i$ sharp in $\bigcup_{i=1}^n V_i$.*

ii) Let φ_i be an **EXor***-derivation of an affine formula B_i from another affine formula A_i sharp in a set of variables V_i for every $1 \leq i \leq n$. Assume that, for every $1 \leq i, j \leq n$ with $i \neq j$, the set of variables* $\mathrm{vars}(A_j \cup B_j)$ *is disjoint from V_i. Then, $\varphi_1 \sqcup \cdots \sqcup \varphi_n$ is an* **EXor***-derivation of $\bigcup_{i=1}^n B_i$ from $\bigcup_{i=1}^n A_i$ sharp in $\bigcup_{i=1}^n V_i$.*

Proof. We show only the first claim; the second one follows by modifying the appropiate notions.

Let β_i be the string of CNF formulae $\left(\bigcup_{j=0}^{i-1} F_j \cup \bigcup_{j=i+1}^n G_j\right) \sqcup \alpha_i$ for every $1 \leq i \leq n$. By Definition 3.31, the fusion $\alpha_1 \sqcup \cdots \sqcup \alpha_n$ is defined as the concatenation $\beta_1 : \ldots : \beta_n$. By the hypothesis that for every $1 \leq i \leq n$, the set of variables $\mathrm{vars}(F_j \cup G_j)$ is disjoint from V, we obtain that:

$$\left(\bigcup_{j=0}^{i-1} F_j \cup \bigcup_{j=i+1}^{n} G_j\right) \cap V = \emptyset$$

Then, Proposition 3.35 shows that β_i is a **Rat**-derivation of the CNF formula $\bigcup_{j=0}^{i-1} F_j \cup \bigcup_{j=i}^n G_j$ from the CNF formula $\bigcup_{j=0}^{i} F_j \cup \bigcup_{j=i+1}^n G_j$ sharp in V_i.

Observe that the final CNF formula in the β_i derivation is the same as the first CNF formula in the β_{i+1} derivation for all $1 \leq i < n$. This implies that the concatenation $\beta_1 : \ldots : \beta_n = \alpha_1 \sqcup \cdots \sqcup \alpha_n$ is a **Rat**-derivation of the last CNF formula in β_n, namely $\bigcup_{j=1}^n G_j$, from the first CNF formula in β_1, which is $\bigcup_{j=1}^n F_j$. Last, this derivation is by Proposition 3.34 sharp in $\bigcup_{i=1}^n V_i$, as we wanted to show. □

Last, we present the overdue results showing that **At**- and **Xor**-derivations can be fused discretionarily. Both results are consequences of regarding derivations in these proof systems as **Rat**- and **EXor**-derivations sharp in $\emptyset$, respectively.

Corollary 3.38 (Fusion for At-derivations): *i) Let α be a* **At***-derivation of a CNF formula G from another CNF formula F, and H be yet another CNF formula. Then $H \sqcup \alpha$ is an* **At***-derivation of $G \cup H$ from $F \cup H$.*

ii) Let α_i be an **At***-derivation of a CNF formula G_i from another CNF formula F_i for every $1 \leq i \leq n$. Then, $\alpha_1 \sqcup \cdots \sqcup \alpha_n$ is an* **At***-derivation of $\bigcup_{i=1}^n G_i$ from $\bigcup_{i=1}^n F_i$.*

Proof. Follows from observing that every **At**-derivation is an **Rat**-derivation sharp in $\emptyset$. □

Corollary 3.39 (Fusion for Xor-derivations): *i) Let φ be a* **Xor***-derivation of an affine formula A' from another affine formula A, and B be yet another affine formula. Then $B \sqcup \alpha$ is an* **Xor***-derivation of $A' \cup B$ from $A \cup B$.*

ii) Let φ_i be an **Xor***-derivation of an affine formula B_i from another affine formula A_i for every $1 \leq i \leq n$. Then, $\alpha_1 \sqcup \cdots \sqcup \alpha_n$ is an* **Xor***-derivation of $\bigcup_{i=1}^{n} B_i$ from $\bigcup_{i=1}^{n} A_i$.*

Proof. Straightforward from considering every **Xor**-derivation as an **EXor**-derivation sharp in $\emptyset$. □

3.4.1 Derivation lengths

We conclude this section by defining an alternative notion of derivation lengths better suited to our purposes. As noted in Section 3.2.1, **Rat**-derivations are recorded in practice as a sequence of introduced and deleted clauses. However, our notion of derivation allows deletion and introduction of multiple clauses in a single step. Furthermore, concatenation of two derivations contains a repetition of a formula, which is a situation that should not add to the length of a derivation. We introduce an adapted notion of derivation length that suits the way **Rat**-derivations are expressed when generating unsatisfiability proofs in SAT solvers, by counting counting how many constraints are introduced or deleted in each inference step. Further issues on the DRAT format used in SAT solving are discussed in Sections 4.2.1 and 8.2.1.

Definition 3.40 (Length of a derivation):
Let $R = \langle L, \models_L, \Rightarrow_R \rangle$ be a proof system, and $\alpha = [F_0, \ldots, F_n]$ be an R-derivation. The length of α, denoted by $\ell(\alpha)$ is given by:

$$\ell(\alpha) = \sum_{i=1}^{n} |F_{i-1} \triangle F_i|$$

Observe that, with the convention that the empty sum is 0, we obtain $\ell([F_0]) = 0$.

We now provide a result that allows the computation of length bounds under composition operations.

Proposition 3.41 (Length bounds under concatenation and fusion):
Let $R = \langle L, \models_L, \Rightarrow_R \rangle$ be a proof system. The following hold:

i) If α is an R-derivation of a formula G from a formula F, and β is an R-derivation of a formula H from G, then $\ell(\alpha : \beta) = \ell(\alpha) + \ell(\beta)$.

ii) If $\alpha = [F_0, \ldots, F_n]$ is an R-derivation and H is a formula, then $\ell(H \sqcup \alpha) \leq \ell(\alpha)$. Furthermore, the equality is reached if and only if H is disjoint from $\bigcup_{i=0}^{n} F_i$.

iii) Let α_i be an R-derivation for all $1 \leq i \leq r$. Then, $\ell(\alpha_1 \sqcup \cdots \sqcup \alpha_r) \leq \sum_{i=1}^{r} \ell(\alpha_i)$.

Proof.

i) Assume that $\alpha = [F_0, \ldots, F_p]$ and $\beta = [G_0, \ldots, G_q]$, where $F_0 = F$, $F_p = G = G_0$ and $G_q = H$. Then, the length of the derivation $\alpha : \beta$ is given by

$$\ell(\alpha : \beta) = \sum_{i=1}^{p} |F_{i-1} \triangle F_i| + |F_p \triangle G_0| + \sum_{i=1}^{p} |G_{i-1} \triangle G_i| = \ell(\alpha) + \ell(\beta)$$

where we have used that $|F_p \triangle G_0| = |G \triangle G| = 0$.

ii) For arbitrary sets S_1, S_2, S_3, we have the following property, which we do not show here:

$(S_1 \cup S_3) \triangle (S_2 \cup S_3) \subseteq S_1 \triangle S_2$. Furthermore, the equality holds if and only if S_3 is disjoint from $S_1 \cup S_2$.

The string $H \sqcup \alpha$ can be expressed as $[H \cup F_0, \ldots, H \cup F_n]$. Observe that by the result above we have

$$\begin{aligned} |(H \cup F_0) \triangle (H \cup F_0)| &\leq |F_0 \triangle F_1| \\ &\vdots \\ |(H \cup F_{n-1}) \triangle (H \cup F_n)| &\leq |F_{n-1} \triangle F_n| \end{aligned} \tag{3.3}$$

and adding the inequalities above yields

$$\ell(H \sqcup \alpha) = \sum_{i=1}^{n} |(H \cup F_{i-1}) \triangle (H \cup F_i)| \leq \sum_{i=1}^{n} |F_{i-1} \triangle F_i| = \ell(\alpha)$$

and the equality holds if and only if equality is reached in every inequality in (3.3), which by the mentioned result is equivalent to H being disjoint from every F_i.

iii) Assume that α_i is an R-derivation of G_i from F_i for every $1 \leq i \leq n$. Then, the fusion $\alpha_1 \sqcup \cdots \sqcup \alpha_n$ is the concatenation of strings $\beta_1, \ldots, \beta_n$ where

$$\beta_i = \left(\bigcup_{j=1}^{i-1} F_i \cup \bigcup_{j=i+1}^{n} G_i \right) \sqcup \alpha_i$$

for every $1 \leq i \leq n$. By claim *(ii)*, we obtain $\ell(\beta_i) \leq \ell(\alpha_i)$, and by claim *(i)* we have:

$$\ell(\alpha_1 \sqcup \cdots \sqcup \alpha_n) = \ell(\beta_1 : \ldots : \beta_n) = \sum_{i=1}^{n} \ell(\beta_i) \leq \sum_{i=1}^{n} \ell(\alpha_i) \qquad \square$$

3.5 Contributions

In this chapter, a formal framework for proof systems has been presented along Section 3.1. This is one of our main contributions: to the best of our knowledge, no satisfactory framework had previously been developed for proof systems where inference rules only preserve satisfiability. Our framework correctly attains this, as shown in Corollaries 3.25 and 3.20.

At this point it is not easy to see the advantage in having such a framework. First of all, it provides tools to compose derivations in a sound way, developed in Section 3.4. Unsatisfiability proofs in a SAT solver are generated in a modular way, i.e. derivation fragments are generated and then concatenated, which poses important problems in non-deep proof systems. This issue has been overlooked in previous literature [MP14, SB06], with the consequent risk for correct generation of unsatisfiability proofs.

One further advantage can be observed in Part II, when the problem of obtaining unsatisfiability proofs for interactive parity reasoning in SAT solvers is reduced to obtaining translations between derivations in different proof systems. Since both proof systems are expressed similarly, translations are much easier to understand and to analyze without incurring in simplifications that may not be supported in practice. Composition operations and length bounds given in Section 3.4 will be unvaluable tools when we are confronted with derivation translations.

The second contribution in this chapter is the development of a proof system that formalizes Gauss-Jordan elimination as a parity reasoning procedure, the **Xor** proof system introduced in Definition 3.21. This formalization allows translation of execution records of the Gauss-Jordan elimination procedure at the level of inferences in the direct translation in Chapter 6 and the lift derivation in Chapter 7.

Last, one minor contribution is Theorem 3.19. In [JHB12], the extension of arbitrary redundancy notions by a look-ahead resolution condition is defined, but cosatisfiability-soundness is not shown for a general extension. Furthermore, our modification of asymmetric tautologies in Definition 3.10 allows avoiding some degenerate cases that violate results in the literature, e.g. [MP14, Lemma 1].

Chapter 4

SAT solving

The *Boolean satisfiability problem* SAT is the decision problem of, given an input CNF formula F, determining whether F is satisfiable or unsatisfiable. The SAT problem has been of utmost importance to the theory of computational complexity, for it was the first problem shown, in an acclaimed paper by Stephen Cook [Coo71], to be *NP-complete* (i.e. every other problem in NP is only polynomially harder than SAT). The complexity class P is widely regarded as containing exactly those problems that are computationally tractable [GJ79], We know that the class NP of languages decidable in polynomial time by *nondeterministic Turing machines* contains every problem in P, yet it is a long-standing open problem whether these two classes are actually equal. Since most problems in NP are regarded as intractable, it has been conjectured that they do not equal; in that case, SAT would be in NP but not in P. This justifies the effort on developing efficient SAT solvers: every other problem in NP can be polynomially reduced to SAT, and therefore they can be regarded as universal solvers for NP problems, using CNF formulae as a programming language of sorts.

An oblique consequence of SAT being in NP is that this introduces an asymmetry when verifying their results. All problems in NP have polynomially verifiable *witnesses* (also known as verifiers [Sip97]) for the "yes" instances. In the case of SAT, an assignment of some variables to 0 or 1 in such a way that any interpretation extending that assignment satisfies the input CNF formula can be verified in polynomial time, e.g. computing the reduct of F under such an assignment. Indeed, almost every state-of-the-art SAT solver provides a satisfying assignment when satisfiability is reported. Such assignments are of great importance to industrial applications in planning [Rin09, GHM+12, KS92].

On the other hand, the class coNP of problems with polynomially verifiable witnesses for the "no" instances [Pap07] has long been conjectured not to equal NP. If this were true, which is unknown as of today, a consequence [Gol08] would be that no NP-complete problem is in coNP. In our setting, this means that, unless NP $=$ coNP, no efficiently verifiable unsatisfiability proofs exist for unsatisfiable instances of the SAT problem. In other words, for a given proof system R, at least one of the following holds:

a) NP $=$ coNP

b) R has no polynomially-sized unsatisfiability proof for a family of unsatisfiable formulae.

c) Inferences by R are not verifiable in polynomial time.

Traditionally, SAT solvers did not attach unsatisfiability proofs to unsatisfiability results. However, the need for correctness of results reported by SAT solvers has gradually led to an evergrowing number of state-of-the-art solvers with proof generation capabilities [Man14a, Bie14, AS14, Soo14]. although several technical and theoretical problems remain unsolved [HB15].

In this chapter, the basics of SAT solvers and unsatisfiability proof generation are presented. Section 4.1 introduces the principles on which standard CDCL-style SAT solvers operate. We describe SAT solvers

from a theoretical perspective as transition systems. Furthermore, a generalized conflict analysis and clause learning environment is described; classical conflict analysis methods are abstracted as an instance of this setting. Unsatisfiability proof generation is discussed in Section 4.2. First, the DRAT proof format for SAT solvers is introduced, and correspondence with **Rat**-derivations is shown. Then, a method to generate unsatisfiability proofs in a modular fashion is described. Last, Section 4.3 reviews the contributions presented in this chapter.

4.1 CDCL-style SAT solvers

One of the main reasons for the vast improvements in SAT solving in the last decades has been the adoption of the *conflict-driven clause learning* (CDCL) approach. In CDCL-style SAT solvers, a search procedure for satisfying assignments is directed by means of *clause learning*.

Definition 4.1 (Partial assignments):
A decision literal *is a decorated literal of the form* $l^\bullet$. *A* (partial) assignment *is a string* $\nu = [l_1, \ldots, l_n]$ *over literals and decision literals such that, for every literal* l *such that one among* $l, \bar{l}, l^\bullet, \bar{l}^\bullet$ *occurs in* ν, *the other three do not occur in* ν. *For a partial assignment* $\nu = [l_1, \ldots, l_n]$, *we define the following:*

- *The* set of literals $\mathrm{lits}(\nu)$ *of* ν *is defined as* $\{l \in \mathbf{Lit} : \text{for some } 1 \leq i \leq n,\ l_i = l \text{ or } l_i = l^\bullet\}$, *i.e. the set of literals disregarding the decoration.*
- *The* set of variables $\mathrm{vars}(\nu)$ *of* ν *is defined as* $\mathrm{vars}(\mathrm{lits}(\nu))$.
- *The* set of atoms $\mathrm{atoms}(\nu)$ *of* ν *is defined as* $\mathrm{atoms}(\mathrm{lits}(\nu))$.
- *The* reduct *of a CNF formula* F *by* ν *is the CNF formula* $F|_{\mathrm{lits}(\nu)}$.
- *The* affine formula $\mathrm{xor}(\nu)$ *generated by* ν *is defined by* $\{\mathrm{xor}(l) : l \in \mathrm{lits}(\nu)\}$.

Note in particular that $\mathrm{lits}(\nu)$ *is a consistent set of literals.*

The execution of a CDCL-style SAT solver is modeled here by a *transition system*. *States* in our transition system are given by pairs $\langle F, \nu \rangle$ where F is a CNF formula and ν is a partial assignment. A state is said to be *satisfied* if the reduct $F|_\nu$ is empty, and *failed* if the unsatisfiable clause $\mathsf{or}(\emptyset)$ occurs in F. Observe that these two conditions are mutually exclusive: whenever the unsatisfiable clause is in F, it occurs in any reduct of F as well. Furthermore, we have the following result:

Proposition 4.2 (Soundness of CDCL-style SAT solvers):
Let F *be a CNF formula. Then,* F *is satisfiable if and only if there is a partial assignment* ν *such that* $\langle F, \nu \rangle$ *is satisfied.*

Proof. We first show the "only if" implication. If F is satisfiable, then let I be an interpretation satisfying F. Enumerate $\mathrm{vars}(F)$ as $\{a_1, \ldots, a_n\}$, and consider literals:

$$l_i = a_i, \quad \text{if } I(a_i) = 1 \qquad\qquad l_i = \neg a_i, \quad \text{if } I(a_i) = 0 \tag{4.1}$$

The partial interpretation ν is given by the string $[l_1, \ldots, l_n]$. Let $L = \mathrm{lits}(\nu) = \{l_1, \ldots, l_n\}$. Assume there is a clause D in the reduct $F|_\nu$. Then, there must be a clause $C \in F$ such that $\mathrm{lits}(C)$ is disjoint from L, and furthermore $D = \mathsf{or}(l \in \mathrm{lits}(C) : \bar{l} \notin L)$. Since I satisfies F, there is a literal l in $\mathrm{lits}(C)$ satisfied by l. Now, $l \in \mathrm{atoms}(F)$, which in particular implies that its underlying variable $\mathrm{var}(l)$ is in $\mathrm{vars}(F) = \{a_1, \ldots, a_n\}$. Hence, for some $1 \leq i \leq n$, either $l = a_i$ or $l = \neg a_i$. It is straightforward to check that, since $I \models l$, then $l = l_i$. But this is in contradiction with the disjointness condition above. By reduction to absurd, we conclude that $F|_\nu$ contains no clause, i.e. $\langle F, \nu \rangle$ is satisfied.

We now show the "if" implication. Let I be an interpretation such that, for every l or $l^\bullet$ occurring in ν, the literal l is satisfied by I. Such an interpretation exists due to the definition of partial assignment. Since

$\langle F, \nu\rangle$	$\xrightarrow[\text{unit}]{}\ \langle F, \nu : [l]\rangle$	iff $\mathsf{or}(l) \in F\big\vert_\nu$
$\langle F, \nu\rangle$	$\xrightarrow[\text{decide}]{}\ \langle F, \nu : [l^\bullet]\rangle$	iff $l \in \text{lits}(F) \setminus \text{lits}(\nu)$
$\langle F, \nu_1 : \nu_2\rangle$	$\xrightarrow[\text{back}]{}\ \langle F, \nu_1\rangle$	
$\langle F, \nu\rangle$	$\xrightarrow[\text{learn}]{}\ \langle F \cup \{\mathcal{C}(\sigma)\}, \nu\rangle$	iff K is an implication graph compatible with F and σ is a shift on K
$\langle F, \nu\rangle$	$\xrightarrow[\text{C-learn}]{}\ \langle F \cup \{\mathcal{C}(\sigma)\}, \nu\rangle$	iff K is an implication graph classically compatible with $\langle F, \nu\rangle$ and σ is a shift on K

Table 4.1: Rules for classical CDCL SAT solving.

the reduct $F\big\vert_\nu$ is empty, then every clause in F contains a literal l such that either l or $l^\bullet$ occurs in ν, hence l is satisfied by I. Therefore, every clause in F is satisfied by I, and we conclude that F is satisfiable. □

Although it would be interesting to show that our transition system is complete (i.e. there is a transition path from any state including an unsatisfiable formula to a failed state), this is not necessary for our goals, since we only aim to model the execution of CDCL-style SAT solvers regardless of whether they are complete or not. A proof can be adapted from other proposals in the literature such as the DPLL System with Learning [NOT05], the Generic CDCL framework [HMPS14] and the Linearized DPLL [Arn09], since our transition system is a variation on them.

4.1.1 The DPLL phase

A standard CDCL algorithm works by alternating two phases. The algorithm is started with state $\langle F, []\rangle$ where F is the input CNF formula. In the first phase, which we refer to as the *DPLL phase* due to its resemblance to the *Davis-Putnam-Loveland-Logemann procedure* [DLL62], *unit propagation* is performed until saturation. That is, if a unit clause $\mathsf{or}(l)$ is found in the reduct $F\big\vert_\nu$, then the literal l is appended to the assignment ν, and this operation is repeated until no more unit clauses are available in the reduct. When this happens, the algorithm *guesses* an unassigned literal l and appends it to the assignment ν decorated by a decision mark as $l^\bullet$. Observe that, throughout the DPLL phase, the CNF formula in the state is preserved and only the assignment is changed. The DPLL phase stops when either of the following happens:

- The current state $\langle F, \nu\rangle$ is satisfied. In this case, as shown by Proposition 4.2, any interpretation extending the assignment ν satisfies F, and then satisfiability is reported.
- The reduct $F\big\vert_\nu$ contains the unsatisfiable clause $\mathsf{or}(\emptyset)$. In this case, we say that a conflict is detected, and the second phase, called *conflict analysis*, is triggered.

Furthermore, during the DPLL phase, the algorithm is allowed to *backtrack* in the assignment stack. That is, the algorithm can drop literals in the end of the assignment ν.

In principle, this system already allows a search procedure able to detect satisfiability whenever the input CNF formula is satisfiable, which guarantees soundness. However, it is not complete: since the CNF formula is preserved, the unsatisfiable clause is never introduced in an unsatisfiable CNF formula. The transition system modeling the DPLL phase corresponds to the union of transition rules $\xrightarrow[\text{unit}]{}$, $\xrightarrow[\text{decide}]{}$ and $\xrightarrow[\text{back}]{}$, shown in Table 4.1.

Example 4.3 (DPLL phase of a CDCL-style SAT solver):
Consider the CNF formula F containing the following clauses:

$$C_1 = \mathsf{or}(\mathsf{p}_0) \qquad C_2 = \mathsf{or}(\neg\mathsf{p}_0, \neg\mathsf{p}_1, \mathsf{p}_2) \qquad C_3 = \mathsf{or}(\neg\mathsf{p}_1, \neg\mathsf{p}_3, \neg\mathsf{p}_4)$$
$$C_4 = \mathsf{or}(\neg\mathsf{p}_1, \mathsf{p}_3, \neg\mathsf{p}_4) \qquad C_5 = \mathsf{or}(\mathsf{p}_1, \mathsf{p}_7) \qquad C_6 = \mathsf{or}(\neg\mathsf{p}_1, \mathsf{p}_4, \mathsf{p}_5)$$

$$C_7 = \mathsf{or}(\neg p_5, p_6) \qquad C_8 = \mathsf{or}(\neg p_5, \neg p_7) \qquad C_9 = \mathsf{or}(\neg p_6, p_7)$$

We execute the DPLL phase of the CDCL algorithm for input F below. The algorithm starts with an empty assignment. If a unit clause occurs in the reduct, the unique literal occurring in it is added to the current assignment using the $\xrightarrow[\text{unit}]{}$ rule. If unit propagation is no longer possible, the algorithm guesses a literal and appends it to the current assignment using the $\xrightarrow[\text{decide}]{}$ rule.

Rule	Assignment ν	Reduct $F\|_\nu$
	$[]$	F
$\xrightarrow[\text{unit}]{}$	$[p_0]$	$\{\mathsf{or}(\neg p_1, p_2)$, $\mathsf{or}(\neg p_1, \neg p_3, \neg p_4)$, $\mathsf{or}(\neg p_1, p_3, \neg p_4)$, $\mathsf{or}(p_1, p_7)$, $\mathsf{or}(\neg p_1, p_4, p_5)$, $\mathsf{or}(\neg p_5, p_6)$, $\mathsf{or}(\neg p_5, \neg p_7)$, $\mathsf{or}(\neg p_6, p_7)\}$
$\xrightarrow[\text{decide}]{}$	$[p_0, p_1^\bullet]$	$\{\mathsf{or}(p_2)$, $\mathsf{or}(\neg p_3, \neg p_4)$, $\mathsf{or}(p_3, \neg p_4)$, $\mathsf{or}(p_4, p_5)$, $\mathsf{or}(\neg p_5, p_6)$, $\mathsf{or}(\neg p_5, \neg p_7)$, $\mathsf{or}(\neg p_6, p_7)\}$
$\xrightarrow[\text{unit}]{}$	$[p_0, p_1^\bullet, p_2]$	$\{\mathsf{or}(\neg p_3, \neg p_4)$, $\mathsf{or}(p_3, \neg p_4)$, $\mathsf{or}(p_4, p_5)$, $\mathsf{or}(\neg p_5, p_6)$, $\mathsf{or}(\neg p_5, \neg p_7)$, $\mathsf{or}(\neg p_6, p_7)\}$
$\xrightarrow[\text{decide}]{}$	$[p_0, p_1^\bullet, p_2, p_3^\bullet]$	$\{\mathsf{or}(\neg p_4)$, $\mathsf{or}(p_4, p_5)$, $\mathsf{or}(\neg p_5, p_6)$, $\mathsf{or}(\neg p_5, \neg p_7)$, $\mathsf{or}(\neg p_6, p_7)\}$
$\xrightarrow[\text{unit}]{}$	$[p_0, p_1^\bullet, p_2, p_3^\bullet, \neg p_4]$	$\{\mathsf{or}(p_5)$, $\mathsf{or}(\neg p_5, p_6)$, $\mathsf{or}(\neg p_5, \neg p_7)$, $\mathsf{or}(\neg p_6, p_7)\}$
$\xrightarrow[\text{unit}]{}$	$[p_0, p_1^\bullet, p_2, p_3^\bullet, \neg p_4, p_5]$	$\{\mathsf{or}(p_6)$, $\mathsf{or}(\neg p_7)$, $\mathsf{or}(\neg p_6, p_7)\}$
$\xrightarrow[\text{unit}]{}$	$[p_0, p_1^\bullet, p_2, p_3^\bullet, \neg p_4, p_5, p_6]$	$\{\mathsf{or}(\neg p_7)$, $\mathsf{or}(p_7)\}$
$\xrightarrow[\text{unit}]{}$	$[p_0, p_1^\bullet, p_2, p_3^\bullet, \neg p_4, p_5, p_6, p_7]$	$\{\mathsf{or}(\emptyset)\}$

At this point, a conflict is detected and the conflict analysis phase is triggered. ■

4.1.2 Implication graphs and clause learning

The second phase is referred to as *conflict analysis*. During conflict analysis, a new clause C entailed by the CNF formula F is derived using an *implication graph* [MMZM01] and added to F. In this Thesis we deal with *parity reasoning-based implication graphs* as proposed by Laitinen, Junttila and Niemelä [Lai14, LJN12, LJN14b], which include substantial differences from *classical implication graphs* used in standard CDCL SAT solving. Considering unsatisfiability proof generation over them requires a generalized notion of implication graphs. We now give a very general schema for implication graphs; those used in classical and parity reasoning-based conflict analysis are defined as instances of this schema.

Our approach is based on considering directly an implication graph constructed in some way from the current state of the SAT solver. Our approach does not depend on how exactly implication graphs are generated. Instead, we simply give a model for general implication graphs and learned clauses, and later consider only implication graphs that are compatible with the state of the SAT solver.

We consider a special symbol $\bot$, called *conflict*, which will be used in our framework as a literal with the following properties:

$$\overline{\bot} = \bot \qquad \mathsf{or}(\bot) = \mathsf{or}(\emptyset) \qquad \mathrm{xor}(\bot) = \mathsf{par}(\top) \tag{4.2}$$

To avoid redefining our syntactical elements and their semantics, it is useful to think of the definitions

above as mere abbreviations. A *pseudoliteral* is either $\bot$ or a literal; a set of pseudoliterals L is *consistent* if $L \setminus \{\bot\}$ is.

Definition 4.4 (Implication graphs and reason clauses):
An implication graph *K is a finite, directed, acyclic graph together with a set of* assumption vertices *$K^\bullet$ satisfying the following properties:*

i) The set of vertices of K is a consistent set of pseudoliterals.

ii) Every assumption vertex $\lambda \in K^\bullet$ is a vertex in K without any predecessors.

iii) $\bot$ is a non-assumption vertex in K without any successors.

For every non-assumption vertex λ in K, we define the reason clause *for λ as the clause:*

$$\mathcal{R}(\lambda) = \mathsf{or}(\lambda) \vee \mathsf{or}(\overline{\mu} : \mu \text{ precedes } \lambda \text{ in } K)$$

We say that K is compatible *with a CNF formula F if $F \models\!\!\!| \mathcal{R}(\lambda)$ for all non-assumption vertices λ in K.*

In our framework, we consider *shifts* over an implication graph, which correspond to the notion of *cut* in the literature [MMZM01]. Every shift defines a clause that can be *learned*, constructed by collecting the complements of precedent literals outside the shift. Theorem 4.8 was shown in a slightly less general setting in [BKS04], and guarantees that clauses learned from shifts are linear resolvents on the set of reason clauses. Therefore, if the implication graph is compatible with a CNF formula F, then learned clauses are semantic consequences of F, and adding them to F preserves semantic equivalence.

Definition 4.5 (Shifts on implication graphs and learned clauses):
Consider an implication graph K. A shift *on K is a nonempty string $\sigma = [\lambda_0, \ldots, \lambda_n]$ of non-decision vertices in K such that:*

i) $\bot = \lambda_0$, and the λ_i are pairwise distinct for $0 \leq i \leq n$.

ii) For all $1 \leq i \leq n$, the vertex λ_i precedes some vertex in $\{\lambda_0, \ldots, \lambda_{i-1}\}$ in K.

Furthermore, the clause learned *from a shift $\sigma = [\lambda_0, \ldots, \lambda_n]$, denoted by $\mathcal{C}(\sigma)$, is the clause:*

$$\mathcal{C}(\sigma) = \mathsf{or}(\overline{\mu} : \mu \in K \setminus \{\lambda_0, \ldots, \lambda_n\} \text{ and } \mu \text{ precedes some vertex in } \{\lambda_0, \ldots, \lambda_n\})$$

We now show the main result that warrants soundness of learned clauses: the sequence of learned clauses from initial segments of a shift is a linear resolution in the set of reason clauses for pseudoliterals in the shift. To attain this result, we first show in Lemma 4.6 that such initial segments of a shift are shifts themselves. Then, Lemma 4.7 shows that extending a shift with a single pseudoliteral yields as learned clause the resolvent of the learned clause from the previous shift with the reason clause for that pseudoliteral. The final result in Theorem 4.8 follows by induction in the length of the shift.

Lemma 4.6 (Subshifts are shifts):
Let $\sigma = [\lambda_0, \ldots, \lambda_n]$ be a shift on an implication graph K. Then, for every $0 \leq i \leq n$, the string $\sigma' = [\lambda_0, \ldots, \lambda_i]$ is a shift on K as well.

Proof. Since σ is a shift in K, all of the λ_j are non-decision vertices in K, in particular $\{\lambda_0, \ldots, \lambda_i\}$. Similarly, condition *(i)* in Definition 4.5 holds straightforward. We now show condition *(ii)*. Let $1 \leq j \leq i$. Since σ is a shift, λ_j has at least one successor in $\{\lambda_0, \ldots, \lambda_{j-1}\}$. Observe that $j - 1 < j \leq i$, hence all these pseudovertices occur in σ' before λ_j, which shows property *(ii)* for λ_j in σ'. □

Lemma 4.7 (Extension of shifts and resolution of learned clauses):
Let $\sigma = [\lambda_0, \ldots, \lambda_n]$ be a shift on an implication graph K. Let λ_{n+1} be a pseudoliteral such that $\sigma' = [\lambda_0, \ldots, \lambda_{n+1}]$

is a shift as well on K. Then, the learned clause $\mathcal{C}(\sigma)$ is resolvable with the reason clause $\mathcal{R}(\lambda_{n+1})$ upon the literal $\overline{\lambda_{n+1}}$ with the learned clause $\mathcal{C}(\sigma')$ as resolvent.

Proof. To show that $\mathcal{C}(\sigma)$ is resolvable with $\mathcal{R}(\lambda_{n+1})$ upon λ_{n+1}, we need to check five properties:

- λ_{n+1} *is a literal* since σ' is a shift, pseudoliterals in σ' (therefore those in σ) are pairwise distinct. Now observe that $\lambda_0 = \bot$, which implies that any pseudoliteral λ_i with $i > 0$ is a literal, in particular λ_{n+1}.
- $\overline{\lambda_{n+1}}$ *occurs in the learned clause* $\mathcal{C}(\sigma)$ by the definition of shift, λ_{n+1} precedes some pseudoliteral among $\lambda_0, \ldots, \lambda_n$ in K. Since elements in σ' are pairwise distinct, $\lambda_{n+1} \in K \setminus \{\lambda_0, \ldots, \lambda_n\}$. Then, λ_{n+1} satisfies the conditions from the definition of learned clauses from shifts so that $\overline{\lambda_{n+1}}$ occurs in $\mathcal{C}(\sigma)$.
- λ_{n+1} *occurs in the reason clause* $\mathcal{R}(\lambda_{n+1})$ this follows straightforward from the definition of reason clauses in Definition 4.4 by the observation above that $\lambda_{n+1} \neq \bot$.
- *If l is a literal other than $\overline{\lambda_{n+1}}$ occurring in $\mathcal{C}(\sigma)$, then $\bar{l}$ does not occur in* $\mathcal{R}(\lambda_{n+1})$ if l occurs in $\mathcal{C}(\sigma)$, then $\bar{l}$ is a vertex from K. Now assume that $\bar{l}$ occurs in $\mathcal{R}(\lambda_{n+1})$. Since $l \neq \overline{\lambda_{n+1}}$, this implies that l is a vertex of K. However, K is a consistent set of pseudoliterals, and since l is a literal, then l and $\bar{l}$ cannot simultaneously be vertices in K, which is a contradiction. We conclude that $\bar{l}$ does not occur in $\mathcal{R}(\lambda_{n+1})$.
- *If l is a literal other than λ_{n+1} occurring in $\mathcal{R}(\lambda_{n+1})$, then $\bar{l}$ does not occur in* $\mathcal{C}(\sigma)$ follows from the same reasoning as in the previous property.

We now need to show that $\mathcal{C}(\sigma) \otimes_{\overline{\lambda_{n+1}}} \mathcal{R}(\lambda_{n+1}) = \mathcal{C}(\sigma')$. This is attained by the following chain of equalities

$$
\begin{aligned}
&\text{lits}(\mathcal{C}(\sigma')) = \\
&\quad = \{\overline{\mu} : \mu \in K \setminus \{\lambda_0, \ldots, \lambda_{n+1}\} \text{ and } \mu \text{ predeces some vertex in } \{\lambda_0, \ldots, \lambda_{n+1}\}\} && = \\
&\quad = \{\overline{\mu} : \mu \in K \setminus \{\lambda_0, \ldots, \lambda_n\} \text{ and } \mu \text{ predeces some vertex in } \{\lambda_0, \ldots, \lambda_{n+1}\}\} \setminus \{\lambda_{n+1}\} && = \\
&\quad = \{\overline{\mu} : \mu \in K \setminus \{\lambda_0, \ldots, \lambda_n\} \text{ and } \mu \text{ predeces some vertex in } \{\lambda_0, \ldots, \lambda_n\}\} \setminus \{\lambda_{n+1}\} \quad \cup \\
&\qquad \cup \quad \{\overline{\mu} : \mu \in K \setminus \{\lambda_0, \ldots, \lambda_n\} \text{ and } \mu \text{ predeces } \lambda_{n+1}\} \setminus \{\lambda_{n+1}\} && = \\
&\qquad = (\text{lits}(\mathcal{C}(\sigma)) \setminus \{\overline{\lambda_{n+1}}\}) \cup (\text{lits}(\mathcal{R}(\lambda_{n+1})) \setminus \{\lambda_{n+1}\}) && = \\
&\qquad = \text{lits}\big(\mathcal{C}(\sigma) \otimes_{\overline{\lambda_{n+1}}} \mathcal{R}(\lambda_{n+1})\big)
\end{aligned}
$$

where we have used that λ_{n+1} does not occur in $\mathcal{C}(\sigma)$. □

Theorem 4.8 (Learned clauses are linear resolvents in reason clauses):
Let $\sigma = [\lambda_0, \ldots, \lambda_n]$ be a shift on an implication graph K. If F is a CNF formula containing reason clauses $\mathcal{R}(\lambda_i)$ for all $0 \leq i \leq n$, then the learned clause $\mathcal{C}(\sigma)$ is a linear resolvent in F.

Proof. Consider the strings of pseudoliterals $\sigma_i = [\lambda_0, \ldots, \lambda_i]$ for $0 \leq i \leq n$, which by Lemma 4.6 are shifts on K as well. Each σ_i gives rise to a learned clause $\mathcal{C}(\sigma_i)$. We show that the string of clauses given by $\xi = [\mathcal{C}(\sigma_0), \ldots, \mathcal{C}(\sigma_n)]$ is a linear resolution in F. The claim then follows straightforward from observing that $\sigma_n = \sigma$.

According to Definition 2.8, two properties need to be verified:

- $\mathcal{C}(\sigma_0) \in F$ *holds* we know that $\sigma_0 = [\bot]$. The reason clause for σ_0 is given by:

$$
\begin{aligned}
\mathcal{R}(\bot) &= \mathsf{or}(\bot) \vee \mathsf{or}(\overline{\mu} : \mu \text{ precedes } \bot \text{ in } K) \\
&= \mathsf{or}(\overline{\mu} : \mu \text{ precedes } \bot \text{ in } K)
\end{aligned}
$$

  Furthermore, the clause learned from the shift σ_0 is:

$$
\mathcal{C}([\bot]) = \mathsf{or}(\overline{\mu} : \mu \in K \setminus \{\bot\} \text{ and } \mu \text{ precedes } \bot)
$$

Observe that, since K is an acyclic graph, $\bot$ cannot precede itself, and hence the two clauses above are the same. We conclude that $\mathcal{C}(\sigma_0) = \mathcal{R}(\bot) \in F$.

- *For every $1 \leq i \leq n$, there is a clause $D \in F$ such that $\mathcal{C}(\sigma_{i-1})$ is resolvable with D upon some literal l with resolvent $\mathcal{C}(\sigma_i)$* observe that $\sigma_i = \sigma_{i-1} : [\lambda_i]$. Hence, Lemma 4.7 applies and we conclude that the learned clause $\mathcal{C}(\sigma_{i-1})$ is resolvable with the reason clause $\mathcal{R}(\lambda_i)$ upon $\overline{\lambda_i}$ with resolvent $\mathcal{C}(\sigma_i)$. Since $\mathcal{R}(\lambda_i) \in F$, the property follows. □

Corollary 4.9 (Entailment-soundness of clause learning):
Let $\sigma = [\lambda_0, \ldots, \lambda_n]$ be a shift on an implication graph K. If K is compatible with a CNF formula F, then $F \models \mathcal{C}(\sigma)$.

Proof. Since K is compatible with F, then F entails every reason clause $\mathcal{R}(\lambda_i)$ for $0 \leq i \leq n$. Therefore, $F \models F' = F \cup \{\mathcal{R}(\lambda_i) : 0 \leq i \leq n\}$. By Theorem 4.8, $\mathcal{C}(\sigma)$ is a linear resolvent in F'. In particular, $F' \models \mathcal{C}(\sigma)$, and the claim follows. □

Corollary 4.9 allows the definition of an additional rule $\xrightarrow[\text{learn}]{}$ that introduces clauses learned from a shift on an implication graph compatible with the current CNF formula, as defined in Table 4.1. The CDCL transition system is defined by the transition relation $\xrightarrow[\text{CDCL}]{}$, which is the union of the rules $\xrightarrow[\text{unit}]{}$, $\xrightarrow[\text{decide}]{}$, $\xrightarrow[\text{back}]{}$ and $\xrightarrow[\text{learn}]{}$.

Before we proceed to explain how this rule is applied in practice, we justify the absence in our formalization of an $\xrightarrow[\text{unsat}]{}$ rule as in [HMPS14, Arn09]. The $\xrightarrow[\text{unsat}]{}$ rule is typically defined as:

$$\langle F, \nu \rangle \xrightarrow[\text{unsat}]{} \mathsf{unsat} \quad \text{if and only if } \nu \text{ does not contain decision literals and } \mathsf{or}(\emptyset) \in F|_\nu$$

In general applications of our framework, assumption literals in the implication graph correspond to decision literals in the partial assignment. When ν does not contain decision literals, then the constructed implication graph typically lacks of assumption literals. In such cases, Corollary 4.10 shows that we can always choose a shift where no pseudoliteral has predecessors outside of the shift. This shift yields as learned clause the unsatisfiable clause $\mathsf{or}(\emptyset)$, and introducing it into the CNF formula provokes the solver state to become failed.

Corollary 4.10 (Completeness of clause learning):
Let K be an implication graph without assumption vertices. Then, there is a shift σ with $\mathcal{C}(\sigma) = \mathsf{or}(\emptyset)$.

Proof. Observe that K is a finite graph, shifts only contain pairwise distinct elements and $[\bot]$ is always a shift on K. These observations imply the existence of a shift $\sigma = [\lambda_0, \ldots, \lambda_n]$ on K of maximal length, i.e. such that $\sigma : \mu$ is not a shift in K for any pseudoliteral μ. Assume that $\mathcal{C}(\sigma) \neq \mathsf{or}(\emptyset)$. Then, there exists a pseudoliteral μ in $K \setminus \{\lambda_0, \ldots, \lambda_n\}$ preceding λ_i for some $0 \leq i \leq n$. This implies that $\sigma : [\mu]$ is a shift as well, which contradicts maximality of σ. By reduction to absurd, $\mathcal{C}(\sigma) = \mathsf{or}(\emptyset)$. □

4.1.3 The conflict analysis phase

In practice, only restrictions of the implication graph framework in Section 4.1.2 are applied to derive clauses. The reason is that checking whether an arbitrary implication graph is compatible with the current CNF formula is in general as hard as deciding satisfiability. The simplest way to ensure this is to require reason clauses to belong to the CNF formula in the state of the solver. Such implication graphs are said here to be *classically compatible* with a state of the solver.

Definition 4.11 (Classically compatible implication graphs):
Let $\langle F, \nu \rangle$ be a state of a CDCL-style SAT solver, i.e. F is a CNF formula and ν is a partial assignment. An implication graph K is said to be classically compatible *with $\langle F, \nu \rangle$ if the following hold:*

i) K contains as vertices exactly the literals occurring in ν without decision decoration, together with the $\bot$ symbol.

ii) Assumption vertices in K are exactly those literals occurring as decision literals in K.

iii) For every non-assumption literal λ, its reason clause $\mathcal{R}(\lambda)$ is included in F.

Corollary 4.12 (Entailment-soundness of classical clause learning):
Let $\langle F, \nu\rangle$ be a state of a CDCL-style SAT solver, K be an implication graph classically compatible with $\langle F, \nu\rangle$ and σ be a shift on K. Then, $F \models \mathcal{C}(\sigma)$.

Proof. Since K is classically compatible with $\langle F, \nu\rangle$, then it is straightforward that K is compatible with F, and the result follows by Corollary 4.9. □

Corollary 4.12 allows a restriction on the $\xrightarrow[\text{learn}]{}$ rule, which we denote by $\xrightarrow[\text{C-learn}]{}$, that can be efficiently applied by requiring the constructed implication graph to be classically compatible with the current state of the solver. The rule $\xrightarrow[\text{C-learn}]{}$ is formally defined in Table 4.1. The classical CDCL transition system is defined by the transition relation $\xrightarrow[\text{C-CDCL}]{}$, which is the union of the rules $\xrightarrow[\text{unit}]{}$, $\xrightarrow[\text{decide}]{}$, $\xrightarrow[\text{back}]{}$ and $\xrightarrow[\text{C-learn}]{}$.

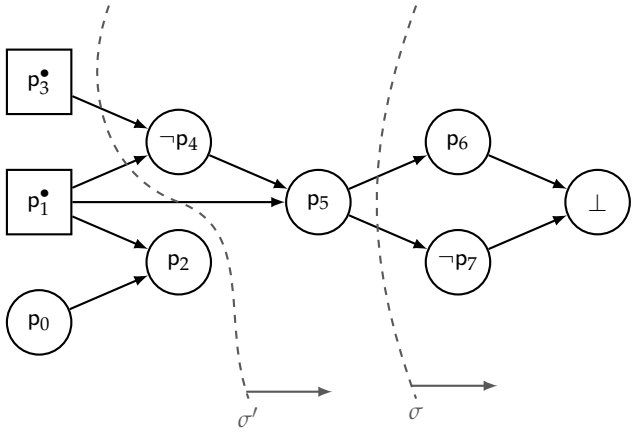
Definition 4.11 does not provide any hint of how classically compatible implication graphs are constructed from a state $\langle F, \nu\rangle$. For the goals of this Thesis, this is largely irrelevant, but for the sake of completeness we informally justify that one can always construct an implication graph upon a conflict. If a literal k is propagated by the $\xrightarrow[\text{unit}]{}$ rule from a state $\langle F, [l_1, \ldots, l_n]\rangle$, then there is a clause $C \in F$ containing k and such that all other literals in C are complements of some of the $l_1, \ldots, l_n$. Let us call such a clause a reason for k. Observe that we only consider in Table 4.1 rules that increment the CNF formula in the solver state. This means that, whenever a chain of $\xrightarrow[\text{CDCL}]{}$ starts in $\langle F, []\rangle$ and ends in a conflict state $\langle F', \nu\rangle$, we can find such reason clauses C_i in F' for every non-decision literal l_i in $\nu = [l_1, \ldots, l_n]$. Furthermore, the unsatisfiable clause $\mathsf{or}(\emptyset)$ occurs in the reduct $F'|_\nu$, so there is a clause $C_\bot$ containing only complements of the $l_1, \ldots, l_n$. We call such a clause a reason for $\bot$. It is straightforward to construct an implication graph classically compatible with $\langle F, \nu\rangle$ by including an edge from a literal l to a pseudoliteral λ if $\bar{l}$ occurs in the reason for λ. Note that acyclicity is guaranteed because, if l_i preceeds l_j, then $i < j$.

In classical CDCL-style SAT solving, new clauses are learned upon a conflict. The goal is to prevent the search procedure from entering in the part of the search space that caused that conflict and redirect it to other fragments of the search space. After a conflict is reached, the solver constructs an implication graph and a shift over it as expounded above, learning a new clause. Then, the solver enters a new DPLL phase, typically started by backtracking.

Example 4.13 (Conflict analysis phase of a CDCL-style SAT solver):
We now continue with the execution from Example 4.3. In the last step there, a conflict was found upon the assignment $\nu = [\mathsf{p}_0, \mathsf{p}_1^\bullet, \mathsf{p}_2, \mathsf{p}_3^\bullet, \neg\mathsf{p}_4, \mathsf{p}_5, \mathsf{p}_6, \mathsf{p}_7]$. A choice for reason clauses is shown below:

Vertex	Reason clause
p_0	$\mathcal{R}(\mathsf{p}_0) = C_1 = \mathsf{or}(\mathsf{p}_0)$
$\mathsf{p}_1^\bullet$	none (decision literal)
p_2	$\mathcal{R}(\mathsf{p}_2) = C_2 = \mathsf{or}(\neg\mathsf{p}_0, \neg\mathsf{p}_1, \mathsf{p}_2)$
$\mathsf{p}_3^\bullet$	none (decision literal)
$\neg\mathsf{p}_4$	$\mathcal{R}(\neg\mathsf{p}_4) = C_3 = \mathsf{or}(\neg\mathsf{p}_1, \neg\mathsf{p}_3, \neg\mathsf{p}_4)$
p_5	$\mathcal{R}(\mathsf{p}_5) = C_6 = \mathsf{or}(\neg\mathsf{p}_1, \mathsf{p}_4, \mathsf{p}_5)$
p_6	$\mathcal{R}(\mathsf{p}_6) = C_7 = \mathsf{or}(\neg\mathsf{p}_5, \mathsf{p}_6)$
$\neg\mathsf{p}_7$	$\mathcal{R}(\neg\mathsf{p}_7) = C_8 = \mathsf{or}(\neg\mathsf{p}_5, \neg\mathsf{p}_7)$
$\bot$	$\mathcal{R}(\bot) = C_9 = \mathsf{or}(\neg\mathsf{p}_6, \mathsf{p}_7)$

The corresponding implication graph looks as follows:



Assumption pseudoliterals are shown in square nodes, and non-assumption pseudoliterals are in circular nodes. Observe that the implication graph shown above is clasically compatible with the last state obtained. Two shifts are shown: $\sigma = [\bot, \neg p_7, p_6]$ with learned clause $\mathcal{C}(\sigma) = \mathsf{or}(p_5)$; and $\sigma' = [\bot, \neg p_7, p_6, p_5, \neg p_4]$ with learned clause $\mathcal{C}(\sigma') = \mathsf{or}(p_1, p_3)$. Learned clauses for both shifts can be obtained as linear resolvents using the reason clauses for pseudoliterals in the shifts:

$$\begin{aligned}
\mathsf{or}(\neg p_6, p_7) &= \mathcal{R}(\bot) \\
\mathsf{or}(\neg p_5, \neg p_6) &= \mathcal{R}(\bot) \otimes_{p_7} \mathcal{R}(\neg p_7) \\
\mathsf{or}(\neg p_5) &= \mathcal{R}(\bot) \otimes_{p_7} \mathcal{R}(\neg p_7) \otimes_{\neg p_6} \mathcal{R}(p_6) && = \mathcal{C}(\sigma) \\
\mathsf{or}(\neg p_1, p_4) &= \mathcal{R}(\bot) \otimes_{p_7} \mathcal{R}(\neg p_7) \otimes_{\neg p_6} \mathcal{R}(p_6) \otimes_{\neg p_5} \mathcal{R}(p_5) \\
\mathsf{or}(\neg p_1, \neg p_3) &= \mathcal{R}(\bot) \otimes_{p_7} \mathcal{R}(\neg p_7) \otimes_{\neg p_6} \mathcal{R}(p_6) \otimes_{\neg p_5} \mathcal{R}(p_5) \otimes_{p_4} \mathcal{R}(\neg p_4) = \mathcal{C}(\sigma')
\end{aligned}$$

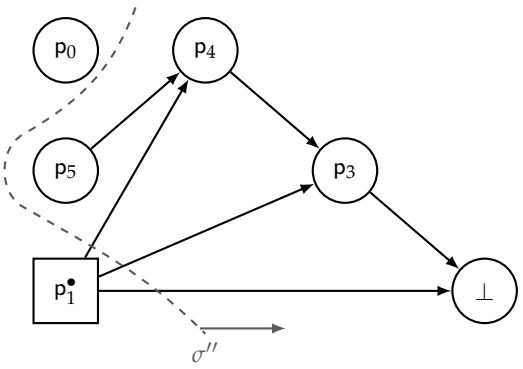
As a consequence, both learned clauses are semantic consequences of F. We apply the $\xrightarrow[\text{C-learn}]{}$ rule to introduce both learned clauses, then backtrack to the first decision literal $p_1^\bullet$ with the $\xrightarrow[\text{back}]{}$ rule.

Rule	Assignment ν	Reduct $F\|_\nu$
$\xrightarrow[\text{C-learn}]{}$	$[p_0, p_1^\bullet, p_2, p_3^\bullet, \neg p_4, p_5, p_6, p_7]$	$\{\mathsf{or}(\emptyset)\}$ *learned:* $\mathsf{or}(\neg p_5)$
$\xrightarrow[\text{C-learn}]{}$	$[p_0, p_1^\bullet, p_2, p_3^\bullet, \neg p_4, p_5, p_6, p_7]$	$\{\mathsf{or}(\emptyset)\}$ *learned:* $\mathsf{or}(\neg p_1, \neg p_3)$
$\xrightarrow[\text{back}]{}$	$[p_0]$	$\{\mathsf{or}(\neg p_1, p_2)$, $\mathsf{or}(\neg p_1, \neg p_3, \neg p_4)$, $\mathsf{or}(\neg p_1, p_3, \neg p_4)$ $\mathsf{or}(p_1, p_7)$, $\mathsf{or}(\neg p_1, p_4, p_5)$, $\mathsf{or}(\neg p_5, p_6)$ $\mathsf{or}(\neg p_5, \neg p_7)$, $\mathsf{or}(\neg p_6, p_7)$, $\mathsf{or}(\neg p_5)$, $\mathsf{or}(\neg p_1, \neg p_3)\}$
$\xrightarrow[\text{unit}]{}$	$[p_0, \neg p_5]$	$\{\mathsf{or}(\neg p_1, p_2)$, $\mathsf{or}(\neg p_1, \neg p_3, \neg p_4)$, $\mathsf{or}(\neg p_1, p_3, \neg p_4)$ $\mathsf{or}(p_1, p_7)$, $\mathsf{or}(\neg p_1, p_4)$, $\mathsf{or}(\neg p_6, p_7)$, $\mathsf{or}(\neg p_1, \neg p_3)\}$
$\xrightarrow[\text{decide}]{}$	$[p_0, \neg p_5, p_1^\bullet]$	$\{\mathsf{or}(p_2)$, $\mathsf{or}(\neg p_3, \neg p_4)$, $\mathsf{or}(p_3, \neg p_4)$, $\mathsf{or}(p_4)$, $\mathsf{or}(\neg p_6, p_7)$, $\mathsf{or}(\neg p_3)\}$
$\xrightarrow[\text{unit}]{}$	$[p_0, \neg p_5, p_1^\bullet, p_4]$	$\{\mathsf{or}(p_2)$, $\mathsf{or}(\neg p_3)$, $\mathsf{or}(p_3)$, $\mathsf{or}(\neg p_6, p_7)$, $\mathsf{or}(\neg p_3)\}$
$\xrightarrow[\text{unit}]{}$	$[p_0, \neg p_5, p_1^\bullet, p_4, p_3]$	$\{\mathsf{or}(p_2)$, $\mathsf{or}(\emptyset)$, $\mathsf{or}(\neg p_6, p_7)\}$

We reach once again a conflict. For the following choice of reason clauses

Vertex	Reason clause
p_0	$\mathcal{R}(\mathsf{p}_0) = \mathsf{or}(\mathsf{p}_0)$
$\neg\mathsf{p}_5$	$\mathcal{R}(\neg\mathsf{p}_5) = \mathsf{or}(\neg\mathsf{p}_5)$
$\mathsf{p}_1^\bullet$	none (decision literal)
$\neg\mathsf{p}_4$	$\mathcal{R}(\mathsf{p}_4) = \mathsf{or}(\neg\mathsf{p}_1, \mathsf{p}_4, \mathsf{p}_5)$
p_3	$\mathcal{R}(\mathsf{p}_3) = \mathsf{or}(\neg\mathsf{p}_1, \mathsf{p}_3, \neg\mathsf{p}_4)$
$\bot$	$\mathcal{R}(\bot) = \mathsf{or}(\neg\mathsf{p}_1, \neg\mathsf{p}_3)$

we obtain the classically compatible implication graph:



The shown shift $\sigma'' = [\bot, \mathsf{p}_3, \mathsf{p}_4, \neg\mathsf{p}_5]$ yields the learned clause $\mathcal{C}(\sigma') = \mathsf{or}(\neg\mathsf{p}_1)$, which can again be obtained as a linear resolvent:

$$\mathcal{R}(\bot) \otimes_{\neg\mathsf{p}_3} \mathcal{R}(\mathsf{p}_3) \otimes_{\neg\mathsf{p}_4} \mathcal{R}(\mathsf{p}_4) \otimes_{\mathsf{p}_5} \mathcal{R}(\neg\mathsf{p}_5) = \mathsf{or}(\neg\mathsf{p}_1) = \mathcal{C}(\sigma')$$

Learning clause $\mathsf{or}(\neg\mathsf{p}_1)$ and backtracking yields:

Rule	Assignment ν	Reduct $F\|_\nu$
$\xrightarrow[\text{C-learn}]{}$	$[\mathsf{p}_0, \neg\mathsf{p}_5, \mathsf{p}_1^\bullet, \mathsf{p}_4, \mathsf{p}_3]$	$\{\mathsf{or}(\mathsf{p}_2)\ ,\ \mathsf{or}(\emptyset)\ ,\ \mathsf{or}(\neg\mathsf{p}_6, \mathsf{p}_7)\}$ *learned:* $\mathsf{or}(\neg\mathsf{p}_1)$
$\xrightarrow[\text{back}]{}$	$[\mathsf{p}_0, \neg\mathsf{p}_5]$	$\{\mathsf{or}(\neg\mathsf{p}_1, \mathsf{p}_2)\ ,\ \mathsf{or}(\neg\mathsf{p}_1, \neg\mathsf{p}_3, \neg\mathsf{p}_4)\ ,\ \mathsf{or}(\neg\mathsf{p}_1, \mathsf{p}_3, \neg\mathsf{p}_4)$ $\mathsf{or}(\mathsf{p}_1, \mathsf{p}_7)\ ,\ \mathsf{or}(\neg\mathsf{p}_1, \mathsf{p}_4)\ ,\ \mathsf{or}(\neg\mathsf{p}_6, \mathsf{p}_7)\ ,\ \mathsf{or}(\neg\mathsf{p}_1, \neg\mathsf{p}_3)\ ,$ $\mathsf{or}(\neg\mathsf{p}_1)\}$
$\xrightarrow[\text{unit}]{}$	$[\mathsf{p}_0, \neg\mathsf{p}_5, \neg\mathsf{p}_1]$	$\{\mathsf{or}(\mathsf{p}_7)\ ,\ \mathsf{or}(\neg\mathsf{p}_6, \mathsf{p}_7)\}$
$\xrightarrow[\text{unit}]{}$	$[\mathsf{p}_0, \neg\mathsf{p}_5, \neg\mathsf{p}_1, \mathsf{p}_7]$	$\emptyset$

The last state $\langle F, [\mathsf{p}_0, \neg\mathsf{p}_5, \neg\mathsf{p}_1, \mathsf{p}_7]\rangle$ is satisfied, and consequently any interpretation I with $I(\mathsf{p}_1) = 0$ and $I(\mathsf{p}_0) = I(\mathsf{p}_5) = I(\mathsf{p}_7) = 1$ satisfies the original CNF formula in Example 4.3. ■

The use of classically compatible implication graphs covers many clause learning techniques, particularly those related to finding useful shifts. In practice, there are many design decisions relevant to the performance of a SAT solver that are conveniently hidden by our abstract framework.

- What decision literals to guess is in practice guided by *branching heuristics* such as *variable state independent decay sum* [MMZ$^+$01] and *phase-saving* [PD07].

- Many reason clauses may exist for a literal during conflict analysis. Furthermore, it may be beneficial to use clauses that do not occur in F as reasons, but are semantic consequences thereof instead. Techniques involving these issues such as *lazy hyper binary resolution* [HJB13], *on-the-fly clause improvement* [HS09] and *local clause minimization* [BKS04] have been proposed.
- In a typical execution of a CDCL SAT solver, implication graphs can be large enough to have a large number of cuts. Much effort has been invested in finding optimal cuts efficiently. The most common-use techniques are based in *unique implication points* [MMZ+01, MMZM01]

4.2 Unsatisfiability proofs

Broad advances in SAT solving came nonetheless at a price. While new methods increased the efficiency of SAT solvers leading to a huge decline in runtime, code complexity rose at a similar pace. Many of the methods used in SAT solving are arduous to implement, which makes it reciprocally easy to introduce errors. Critical bugs are often easily detected, but the most subtle ones sometimes make it unnoticed to released versions.

False positives, i.e. unsatisfiable formulae that are reported as satisfiable, are unproblematic. Virtually every SAT solver provides a partial interpretation satisfying the formula. Had the solver given a wrong answer, incorrectness can be exposed by checking that the interpretation does not satisfy the formula. False negatives, i.e. satisfiable formulae that are reported as unsatisfiable, are harder to detect, since the CDCL approach does not provide a witness for unsatisfiability.

One possible approach to increase the reliability of reported results is to apply *fuzzing techniques* [BLB10], which test the solver in randomly generated instances. Although this method is effective in many cases, it is not complete, since that would require checking all possible instances of the SAT problem. One further approach is to use *software verification* methods to certify correctness of SAT solvers [OSOC12, Mar09]. However, this is too often inconvenient: the solver would need to be verified every time a modification is done, and frequently solvers are large enough for verification to require intensive (and costly) work or even be unfeasible.

The approach this Thesis is concerned with consists on generating an *unsatisfiability proof* every time the solver reports an instance as unsatisfiable [HB15]. To further increase reliability, unsatisfiability proofs can be checked with tools certified by software verification [WHJ13, WHJ14, HJW14]. This approach has been adopted by the International SAT Competition [BDHJ14], and since 2013 all solvers participating in the unsatisfiability track are required to emit unsatisfiability proofs [1]. Apart from certification of results reported by SAT solvers, unsatisfiability proofs have been found useful in minimal unsatisfiable set extraction [BHM14] and interpolant generation [WWLH14].

Resolution proofs and clausal proofs Unsatisfiability proofs emitted by SAT solvers are classified into resolution proofs and clausal proofs. In *resolution proofs* [ZM03], every clause along a derivation expressed in some variant of resolution is listed together with its antecedents, i.e. previosly obtained clauses that justify the derivation of that clause. Resolution proofs are fast to check yet long [HB15], e.g. a linear resolvent would need to be accompanied by the full linear resolution sequence. This is utterly relevant, since verifying such unsatisfiability proofs may involve memory-related issues. Furthermore, generation of resolution proofs requires substantial modifications in solvers [HJW14].

Clausal proofs [GN03] are an alternative approach where antecedents are not given. Clausal proofs are much shorter than resolution proofs, but also harder to verify [HJW14]. One further advantage of clausal proofs is that generation only requires small changes in the code of a solver. For these reasons,

[1] See `http://www.satcompetition.org/2014/certunsat.shtml` .

clausal proofs are generally preferred, and the vast majority of state-of-the-art SAT solvers with proof generation engines produce clausal proofs, including `Riss` [Man14a], `Lingeling` [Bie14], `Glucose` [AS14], `CryptoMiniSAT` [Soo14]. Furthermore, efficient proof checkers based on verifying proofs in reverse order such as `DRAT-trim` [WHJ14] have been developed.

4.2.1 DRAT representations of Rat-derivations

The current paradigm for unsatisfiability proofs is the *Delete Resolution Asymmetric Tautologies (DRAT)* format [WHJ13, WHJ14], corresponding to the **Rat** proof system from Definition 3.18 (page 30). The DRAT proof format has been found very proficient in expressing proofs for several satisfiability-preserving techniques in SAT solving. Proofs for *resolution asymmetric subsumption*, *pure literal elimination*, *blocked clause addition*, *(partial) variable elimination*, [JHB12], *extended resolution*, *vivification*, *common implication* and *bounded variable addition* with either OR or XOR gates [MP14], as well as several *clause elimination procedures* [HJB10] have been proposed within the DRAT proof format.

Proofs in the DRAT format are given as a string of clauses labeled by either D or I, which stand for *delete* and *introduce* respectively. In this work we use a slight variant of the DRAT format, where we consider an additional label N standing for *neglect*. When necessary, we denote clauses C labeled by x as C^x.

Intuitively, DRAT format proofs are compact representations of **Rat**-derivations obtained by recording the changes in the CNF formula along the derivation. That is, clauses removed from the **Rat**-derivation occur labeled by D in their DRAT representation; reciprocally, introduced clauses occur labeled by I. In this way, DRAT proofs are clausal proofs, since they do not record antecedents.

Definition 4.14 (Strings of CNF formulae derived by strings of labeled clauses):
Consider a string of labeled clauses $\varepsilon = [C_1, \ldots, C_n]$, and a CNF formula F. The string derived *by ε from F, denoted by $F * \varepsilon$ is a string $[G_0, \ldots, G_n]$ of CNF formulae such that:*

i) $G_0 = F$.

ii) For every $1 \leq i \leq n$, *if the clause* C_i *in* ε *is labeled by* D, *then* $F_i = F_{i-1} \setminus \{C_i\}$.

iii) For every $1 \leq i \leq n$, *if the clause* C_i *in* ε *is labeled by* I, *then* $F_i = F_{i-1} \cup \{C_i\}$.

iv) For every $1 \leq i \leq n$, *if the clause* C_i *in* ε *is labeled by* N, *then* $F_i = F_{i-1}$.

Definition 4.15 (DRAT representations of Rat-derivations):
Let $\alpha = [F_0, \ldots, F_n]$ be a **Rat***-derivation of F_n from F_0 sharp in V. Let ε be a string of labeled clauses and F be a CNF formula. $\langle F, \varepsilon \rangle$ is said to be* a DRAT representation *of α sharp in V if there are strings of labeled clauses $\varepsilon_1, \ldots, \varepsilon_n$ such that the following hold:*

i) $\varepsilon = \varepsilon_1 : \ldots : \varepsilon_n$

ii) $F = F_0$.

iii) For all $1 \leq i \leq n$, *the string of CNF formulae* $F_{i-1} * \varepsilon_i$ *is a* **Rat***-derivation of* F_i *from* F_{i-1} *sharp in* V.

The notion of DRAT representation draws a correspondence between sequences of labeled clauses and **Rat**-derivations. Furthermore, DRAT representations justify our somewhat odd choice for a length measure of derivations in Definition 3.40: the length of a derivation is precisely the total number of insertions and deletions along it.

Example 4.16 (DRAT representations of Rat-derivations):
Consider clauses:

$$C_1 = \mathsf{or}(\mathsf{p}_1, \mathsf{p}_2) \qquad C_2 = \mathsf{or}(\neg\mathsf{p}_1, \mathsf{p}_3) \qquad C_3 = \mathsf{or}(\neg\mathsf{p}_2, \mathsf{p}_4)$$
$$C_4 = \mathsf{or}(\neg\mathsf{p}_1, \neg\mathsf{p}_2) \qquad C_5 = \mathsf{or}(\neg\mathsf{p}_1, \neg\mathsf{p}_2) \qquad C_6 = \mathsf{or}(\mathsf{p}_1, \neg\mathsf{p}_2)$$

and CNF formulae:

$$F_0 = \{C_1, C_2, C_3, C_4\} \quad F_1 = \{C_1, C_2, C_3, C_4, C_5\} \quad F_2 = \{C_2, C_3, C_4, C_5\} \quad F_3 = \{C_2, C_3, C_4, C_5, C_6\}$$

Since C_5 is an asymmetric tautology in F_0 and C_6 is a resolution asymmetric tautology in F_3, we conclude that the string of CNF formulae $\alpha = [F_0, F_1, F_2, F_3]$ is a **Rat**-derivation of F_3 from F_0. Furthermore, if we define the string of labeled clauses $\varepsilon = [C_5^{\mathsf{I}}, C_1^{\mathsf{D}}, C_3^{\mathsf{N}}, C_6^{\mathsf{I}}]$, then $\langle F_0, \varepsilon\rangle$ is a DRAT representation of α. To verify this, let us find a suitable decomposition of ε as in Definition 4.15:

$$\begin{array}{ccc} \varepsilon_1 = [C_5^{\mathsf{I}}] & \varepsilon_2 = [C_1^{\mathsf{D}}, C_3^{\mathsf{N}}] & \varepsilon_3 = [C_6^{\mathsf{I}}] \\ F_0 * \varepsilon_1 = [F_0, F_1] & F_1 * \varepsilon_2 = [F_1, F_2, F_2] & F_2 * \varepsilon_3 = [F_2, F_3] \end{array}$$

Note that the $F_{i-1} * \varepsilon_i$ are **Rat**-derivations of F_i from F_{i-1} for all $1 \leq i \leq 3$. ■

We need to ensure that our definition of DRAT representation is equivalent to that of **Rat**-derivations. To justify this, we show that the string derived by every DRAT representation of a **Rat**-derivation of G from F is again a **Rat**-derivation of G from F. Reciprocally, for every **Rat**-derivation α, a DRAT representation of α can be constructed.

Proposition 4.17 (DRAT representations correspond to represented Rat-derivations):
Let $\alpha = [F_0, \ldots, F_n]$ be a **Rat***-derivation of F_n from F_0. Let $\langle F_0, \varepsilon\rangle$ be a DRAT representation of α sharp in a set of variables V. Then, $F_0 * \varepsilon$ is a* **Rat***-derivation of F_n from F_0 sharp in V.*

Proof. Since $\langle F_0, \varepsilon\rangle$ is a DRAT representation of α sharp in V, the string ε of labeled clauses can be decomposed as $\varepsilon_1 : \ldots : \varepsilon_n$ with the properties in Definition 4.15. Then we find that for every $1 \leq i \leq n$, the derived string $F_{i-1} * \varepsilon$ is a **Rat**-derivation of F_i from F_{i-1} sharp in V. The Proposition is a consequence from the following claim by particularizing for $i = n$:

For all $0 \leq i \leq n$, the string $F_0 * (\varepsilon_1 : \ldots : \varepsilon_i)$ is a **Rat**-derivation of F_i from F_0 sharp in V.

We show the claim by induction on i. For $i = 0$, we obtain $F_0 * [] = [F_0]$, and the claim follows straightforward. Now assume the claim to hold for some $0 \leq i < n$, and let us show it for $i + 1$. Let us write the derived strings for i and $i + 1$ as follows:

$$F_0 * (\varepsilon_1 : \ldots : \varepsilon_i) = [G_0, \ldots, G_p] \tag{4.3}$$

$$F_0 * (\varepsilon_1 : \ldots : \varepsilon_{i+1}) = [G'_0, \ldots, G'_q] \tag{4.4}$$

From Definition 4.14 it is clear that $q \geq p$ and $G_j = G'_j$ for all $0 \leq j \leq p$. Since (4.3) is a **Rat**-derivation of F_i from F_0, then $G'_{j-1} = G_{j-1} \Rightarrow_{\mathbf{Rat}} G_j = G'_j$ for every $1 \leq j \leq p$, and furthermore $G_p = G'_p = F_i$. The latter implies that $[G'_p, \ldots, G'_q] = F_i * \varepsilon_{i+1}$, which is a **Rat**-derivation of F_{i+1} from F_i, so the inference $G'_{j-1} \Rightarrow_{\mathbf{Rat}} G'_j$ holds for every $p < j \leq q$ and $G_q = F_{i+1}$. This shows that (4.4) is a **Rat**-derivation of F_{i+1} from F_i. Similarly, sharpness in V of (4.4) follows from sharpness in V of (4.3) and $F_i * \varepsilon_{i+1}$. Then, the claim for $i + 1$ holds. By induction, the full claim, and consequently the Proposition, follow. □

Proposition 4.18 (Existence of DRAT representations):
Let $\alpha = [F_0, \ldots, F_n]$ be a **Rat***-derivation of F_n from F_0 sharp in a set of variables V. Observe that, for every $1 \leq i \leq n$, either $F_{i-1} \subseteq F_i$ or $F_i \subseteq F_{i-1}$. Consider a string ε_i of labeled clauses for every $1 \leq i \leq n$ such that:*

i) If $F_i \subseteq F_{i-1}$, then ε_i is an enumeration of $F_{i-1} \setminus F_i$ where all clauses are labeled by D.

ii) If $F_{i-1} \subseteq F_i$, then ε_i is an enumeration of $F_i \setminus F_{i-1}$ where all clauses are labeled by I.

In the case when $F_i = F_{i-1}$, both preconditions overlap, but they yield the same ε_i, namely the empty string. Define

ε as the concatenation $\varepsilon_1 : \ldots : \varepsilon_n$. Then, $\langle F_0, \varepsilon\rangle$ is a DRAT representation of α sharp in V.

Proof. Conditions *(i)* and *(ii)* in Definition 4.15 hold straightforward. We show now condition *(iii)*. We distinguish three cases:

- If $F_i \subseteq F_{i-1}$, then $\varepsilon_i = [C_1^{\mathsf{D}}, \ldots, C_p^{\mathsf{D}}]$ enumerates $F_{i-1} \setminus F_i$. The string derived by ε_i from F_0 is then given by $F_{i-1} * \varepsilon_i = [G_0, \ldots, G_p]$ where $G_j = F_{i-1} \setminus \{C_1, \ldots, C_j\}$ for every $0 \leq j \leq p$. Then, we have $G_0 \supseteq G_1 \supseteq \cdots \supseteq G_p$, and by clause deletion we obtain

$$G_0 \Rightarrow_{\mathbf{Rat}} G_1 \Rightarrow_{\mathbf{Rat}} \cdots \Rightarrow_{\mathbf{Rat}} G_{p-1} \Rightarrow_{\mathbf{Rat}} G_p \tag{4.5}$$

  and then $F_{i-1} * \varepsilon_i$ is a **Rat**-derivation of $G_p = F_{i-1} \setminus (F_{i-1} \setminus F_i) = F_i$ from F_{i-1}. Furthermore, all inferences are clause deletions, which are **At** inferences, so in particular F_{i-1} is sharp in $\emptyset$, hence in V.

- If $F_{i-1} \subseteq F_i$ and $F_i \setminus F_{i-1} = \{C_1, \ldots, C_p\}$ is a set of asymmetric tautologies in F_{i-1}, then ε_i is $[C_1^{\mathsf{I}}, \ldots, C_p^{\mathsf{I}}]$. The string derived by ε_i from F_0 is then given by $F_{i-1} * \varepsilon_i = [G_0, \ldots, G_p]$ where $G_j = F_{i-1} \cup \{C_1, \ldots, C_j\}$ for every $0 \leq j \leq p$. Observe that, by Lemma 3.12, every C_j is an asymmetric tautology in $F_{i-1} \cup \{C_1, \ldots, C_{j-1}\} = G_j$ for $1 \leq j \leq p$. By asymmetric tautology introduction, we obtain (4.5) again, so $F_{i-1} * \varepsilon_i$ is a **Rat**-derivation of $G_p = F_{i-1} \cup (F_i \setminus F_{i-1}) = F_i$ from F_{i-1}. Sharpness in V follows similarly to the previous case, since asymmetric tautology introductions are **At** inferences.

- If $F_{i-1} \subseteq F_i$ and $F_i \setminus F_{i-1} = \{C_1, \ldots, C_p\}$ is an unresolvable set of resolution asymmetric tautologies in F_{i-1}, then $\varepsilon_i = [C_1^{\mathsf{I}}, \ldots, C_p^{\mathsf{I}}]$. If we define CNF formulae $G_j = F_{i-1} \cup \{C_1, \ldots, C_j\}$ for every $0 \leq j \leq p$ and, we obtain $F_{i-1} * \varepsilon_i = [G_0, \ldots, G_p]$. We show that C_j is a resolution asymmetric tautology in G_{j-1} for all $0 \leq j \leq p$. Assume there is a clause D in G_{j-1} such that C_j is resolvable with D upon a literal l. Since $\{C_1, \ldots, C_p\}$ is unresolvable, then $D \in F_{i-1}$. Because C_j is a resolution asymmetric tautology in F_{i-1} upon a literal l with $\mathrm{var}(l) \in V$, due to sharpness of α, we conclude that $C_j \otimes_l D$ is an asymmetric tautology in F_{i-1}. By Lemma 3.12, $C_j \otimes_l$ is an asymmetric tautology in G_{j-1} as well, which makes C_j a resolution asymmetric tautology in G_{j-1}, and we obtain $G_{j-1} \Rightarrow_{\mathbf{Rat}} G_j$, so $F_{i-1} * \varepsilon_i$ is a **Rat**-derivation of $G_p = F_i$ from F_{i-1}. Furthermore, it is sharp in V, because the literals l considered above have underlying variables $\mathrm{var}(l)$ in V. □

Propositions 4.17 and 4.18 show that there is no significant difference between using **Rat**-derivations or DRAT proofs. For coherence with our framework for proof systems, throughout this Thesis we mainly deal with **Rat**-derivations. Some additional remarks on the practical generation of DRAT proofs are provided in Section 8.2.1.

4.2.2 Generating unsatisfiability proofs in CDCL-style SAT solvers

The formal framework for SAT solving presented in Section 4.1 allows the generation of unsatisfiability proofs in a modular way. Given an run of a SAT solver, we can compose partial **Rat**-derivations for every computation step into a full unsatisfiability proof, i.e. a **Rat**-derivation of $\{\mathsf{or}(\emptyset)\}$ from the input CNF formula F, by concatenation. In particular, we consider a run of a CDCL-style SAT solver upon an input CNF formula F finally reporting unsatisfiability, that is, a $\xrightarrow[\text{CDCL}]{}$-chain

$$\langle F, []\rangle = \langle F_0, \nu_0\rangle \xrightarrow[\text{CDCL}]{} \langle F_1, \nu_1\rangle \xrightarrow[\text{CDCL}]{} \cdots \xrightarrow[\text{CDCL}]{} \langle F_{n-1}, \nu_{n-1}\rangle \xrightarrow[\text{CDCL}]{} \langle F_n, \nu_n\rangle$$

where $\langle F_n, \nu_n\rangle$ is failed, or equivalently, $\mathsf{or}(\emptyset) \in F_n$. If **Rat**-derivations α_i of F_i from F_{i-1} are given for every $1 \leq i \leq n$, then by Proposition 3.6 an unsatisfiability proof for F can be generated as a **Rat**-derivation α defined by $\alpha = \alpha_1 : \ldots : \alpha_n : [F_n, \{\mathsf{or}(\emptyset)\}]$.

In our transition system, the only rules that modify the CNF formula in the state of the solver are $\xrightarrow[\text{learn}]{}$ and its variation $\xrightarrow[\text{C-learn}]{}$. Hence, suitable **Rat**-derivations for all other inference rules $\xrightarrow[\text{unit}]{}$, $\xrightarrow[\text{decide}]{}$ and $\xrightarrow[\text{back}]{}$ are given by $\alpha_i = [F_i] = [F_{i-1}]$. Derivations for the $\xrightarrow[\text{learn}]{}$ rule can be constructed as shown in Proposition 4.19, provided that derivations for reason clauses are given.

Proposition 4.19 (Unsatisfiability proofs for the $\xrightarrow[\text{learn}]{}$ rule):
*Let K be an implication graph compatible with a CNF formula F, and $\sigma = [\lambda_0, \ldots, \lambda_n]$ be a shift in K. Define the CNF formula $G = F \cup \{\mathcal{R}(\lambda_i) : 0 \leq i \leq n\}$. For every $0 \leq i \leq n$, let α_i be a **Rat**-derivation of $F \cup \{\mathcal{R}(\lambda_i)\}$ from F sharp in a set of variables V_i. If $\text{vars}(G)$ is disjoint from V_i for all $0 \leq i \leq n$, then the string of CNF formulae*

$$\alpha = (\alpha_0 \sqcup \cdots \sqcup \alpha_n) : [G\ ,\ G \cup \{\mathcal{C}(\sigma)\}\ ,\ F \cup \{\mathcal{C}(\sigma)\}]$$

*is a **Rat**-derivation of $F \cup \{\mathcal{C}(\sigma)\}$ from F.*

Proof. Observe that both premises and conclusions of every α_i are included in $\text{vars}(G)$. Therefore, the conditions in Proposition 3.37 hold, and $\alpha_0 \sqcup \cdots \sqcup \alpha_n$ is a **Rat**-derivation of $\bigcup_{i=1}^{n}(F \cup \{\mathcal{R}(\lambda_i)\}) = G$ from $\bigcup_{i=1}^{n} F = F$. Checking that the string $[G\ ,\ G \cup \{\mathcal{C}(\sigma)\}\ ,\ F \cup \{\mathcal{C}(\sigma)\}]$ is a **Rat**-derivation is straightforward: the first inference holds by clause deletion, and the second by asymmetric tautology introduction, due to Theorem 4.8 and Proposition 3.16. Therefore, their concatenation α is a **Rat**-derivation as in the claim. □

In the particular setting of classical CDCL-style SAT solving, the $\xrightarrow[\text{learn}]{}$ rule is restricted to the $\xrightarrow[\text{C-learn}]{}$ rule where learned clauses occur in the CNF formula of the predecessor state. The following result allows the construction of unsatisfiability proofs for classical CDCL-style SAT solving by providing **Rat**-derivations of F_i from F_{i-1} for the cases when $\langle F_{i-1}, \nu_{i-1}\rangle \xrightarrow[\text{C-learn}]{} \langle F_i, \nu_i\rangle$. Although these can be constructed by giving trivial **Rat**-derivations of F_{i-1} from F_{i-1} as the derivations α_i in Proposition 4.19, we can simply use one asymmetric tautology introduction inference to obtain F_i.

Corollary 4.20 (Unsatisfiability proofs for the $\xrightarrow[\text{C-learn}]{}$ rule):
*Let K be an implication graph classically compatible with a state $\langle F, \nu\rangle$, and σ be a shift in K. Then, the string of CNF formulae $\alpha = [F, F \cup \{\mathcal{C}(\sigma)\}]$ is a **Rat**-derivation of $F \cup \{\mathcal{C}(\sigma)\}$ from F.*

Proof. Since all reason clauses for non-assumption vertices in K are in F, the result follows straightforward from Proposition 3.16 and Theorem 4.8. □

4.3 Contributions

The main contribution in this chapter is the generalized conflict analysis and clause learning setting described in Section 4.1.2. As explained later in Section 5.3, this setting is able to describe the parity reasoning-based conflict analysis methods proposed by Laitinen, Junttila and Niemelä [Lai14, LJN12, LJN14b]. We have furthermore proposed a general schema to generate unsatisfiability proofs for SAT solving in a modular way using Proposition 4.19. Under this schema, generation of unsatisfiability proofs is reduced to generate proofs for reason clauses used in conflict analysis, regardless of how conflict graphs are generated in practice.

These two contributions were developed in settings as general as possible. The goal is to give a framework that is useful not only for proof generation in SAT solvers with integrated parity reasoning, but also for SAT solvers coupled with other kinds of specialized reasoning procedures. In particular, we are interested to extend in the future our ideas to include cardinality resolution [RM09], which has been proposed as an open question in the literature [HB15, SB06].

Chapter 5

Parity reasoning in SAT solving

However vast recent advances in SAT solving have been, intrinsic limitations of the underlying proof system circumscribe their reach. Knot Pipatsrisawat and Adnan Darwiche [PD11] recently showed that, from a proof complexity perspective, modern CDCL SAT solvers are as powerful as the resolution proof system. This has deep consequences for SAT solving, since it is known that resolution yields exponential length proofs on some fragments. Armin Haken showed [Hak85] in 1985 that formulae encoding the pigeonhole principle can only be refuted by resolution using proofs that are exponential on the size of the formulae. A similar result was achieved two years later [Urq87] by Alasdair Urquhart for formulae encoding parity constraints. Performance of CDCL SAT solvers cannot thus be expected to be high in these fragments.

SAT solvers integrating specialized reasoning into a CDCL-style procedure gained popularity in the last years. This thesis is mainly concerned with the integration of polynomial procedures for parity reasoning, such as Gauss-Jordan elimination, into CDCL-style procedures. The proof system underlying Gauss-Jordan elimination is the **Xor** proof system presented in Section 3.3. Integration of parity reasoning in SAT solvers is not covered by the classical CDCL transition rules presented in Section 4.1. The *XOR-CDCL transition* system modeling SAT solvers with integrated parity reasoning adds three new rules:

1. *Affine formula simplification* applies Gauss-Jordan elimination to encoded affine formulae occurring in the CNF formula in the current state. Simplified affine formulae are then encoded back into semantically equivalent CNF formulae, which replace the original encoded affine formula.
2. *Propagation of literals entailed by affine formulae* applies Gauss-Jordan elimination to encoded affine formulae using the current literals in the partial assignment. As shown in Section 5.2, Gauss-Jordan elimination is then able to detect entailed literals that are not propagated by the $\underset{\text{unit}}{\longrightarrow}$ rule, effectively reducing the search space to be explored by guessing literals.
3. *Conflict detection and clause learning using reason parity constraints* allows to learn clauses even if literals in the partial assignment were propagated using method 2. In general, it is not possible to construct a classically compatible implication graph. However, it is possible to extend this notion within the clause learning schema from Section 4.1.2 to learn clauses entailed by the CNF formula in the solver state.

Since new rules are proposed, unsatisfiability proofs for these rules must be generated. Technique 2 is straightforward, since the CNF formula is preserved. For technique 1, a translation of **Xor**-derivations expressing Gauss-Jordan elimination into **Rat**-derivations of the introduced encoded affine formula would provide a proof. In a less obvious way, such a translation can provide derivations for learned clauses in technique 3 as well, namely by deriving reason clauses and then applying Proposition 4.19.

This chapter is structured as follows. First, encodings of affine formulae into CNF formulae are introduced in Section 5.1. Encodings provide the connection between CNF and affine formulae, which so far

in this Thesis have been tackled separately. Section 5.2 is devoted to introduce the Gauss-Jordan elimination procedure, and to show the entailment and conflict detection properties that makes it useful for SAT solving. In Section 5.3 discusses proposed methods to apply parity reasoning to SAT solving. There, we define the XOR-CDCL transition system, which extends the Classic CDCL transition system by the three rules explained above. Finally, Section 5.4 reviews the contributions in this chapter.

5.1 Encodings of parity constraints into CNF formulae

The idea of applying parity reasoning to SAT solving relies in the fact that the expressivity of affine formulae is strictly lower than that of CNF formulae. In particular, parity constraints can be *encoded* into CNF formulae in ways that preserve their semantics in some specific sense. Parity reasoning procedures can then be applied to affine formulae encoded in the CNF formula whose satisfiability the SAT solver is trying to decide. In this section we present two encodings of parity constraints, based on those presented in [GK13].

The simplest of such encodings is the *direct encoding*. The advantage of the direct encoding over other encodings is that it preserves semantic equivalence: Boolean interpretations satisfying the direct encoding of an affine formula A are exactly those satisfying A itself. However, the direct encoding of a parity constraint X contains exponentially many clauses in the size of X, which is detrimental to the performance of a SAT solver operating over direct encodings of affine formulae.

Definition 5.1 (Direct encoding of parity constraints):
Let X be a parity constraint. Its direct encoding $\mathcal{D}(X)$ *is defined as the following CNF formula:*

$$\mathcal{D}(X) = \{\mathsf{nand}(V) \vee \mathsf{or}(\mathrm{vars}(X) \setminus V) : V \subseteq \mathrm{vars}(X) \text{ and } |V| \approx \mathrm{pol}(X) + 1\}$$

We further define the direct encoding of an affine formula A as the CNF formula $\mathcal{D}(A) = \bigcup_{X \in A} \mathcal{D}(X)$.

The definition above can be rephrased as follows: the clauses in the direct encoding of a parity constraint X with $\mathrm{vars}(X) = \{a_1, \ldots, a_n\}$ are those containing exactly literals $l_1, \ldots, l_n$ with $\mathrm{var}(l_i) = a_i$, where the number of negative l_i is even if $\top \in \mathrm{vars}^\star(X)$, and odd otherwise.

Proposition 5.2 ($\mathcal{D}(X)$ is a CNF encoding of X):
Let X be a parity constraint. Then, $F \equiv \mathcal{D}(X)$.

Proof. To show this, we need to prove that any interpretation I satisfies X if and only if it satisfies $\mathcal{D}(X)$. We first show the "only if" implication. Let I be an interpretation satisfying X, and V any subset of the variables in X with $|V| \approx \mathrm{pol}(X) + 1$; or, equivalently, $|V| \not\approx \mathrm{pol}(X)$. Assume that I does not satisfy the clause $\mathsf{nand}(V) \vee \mathsf{or}(\mathrm{vars}(X) \setminus V)$. Then, I maps to 1 every variable in V and to 0 every variable in $\mathrm{vars}(X) \setminus V$. That means that I satisfies exactly $|V|$ variables in X. However, since I satisfies X, then the number of variables in $\mathrm{vars}(X)$ satisfied by I is congruent to $\mathrm{pol}(X)$. This is a contradiction because $|V| \not\approx \mathrm{pol}(X)$. By reduction to absurd, I satisfies $\mathsf{nand}(V) \vee \mathsf{or}(\mathrm{vars}(X) \setminus V)$ for every V with the properties in the definition of $\mathcal{D}(X)$, so $I \models \mathcal{D}(X)$. This shows $X \models \mathcal{D}(X)$.

We now show the "if" implication. Let I be an interpretation satisfying $\mathcal{D}(X)$, and assume $I \not\models X$. Let us denote by V the set of variables occurring in X satisfied by I. Then, $|V| \not\approx \mathrm{pol}(X)$, since otherwise I would satisfy X. Modulo 2, this means that $|V| \approx \mathrm{pol}(X) + 1$, so the clause $C = \mathsf{nand}(V) \vee \mathsf{or}(\mathrm{vars}(X) \setminus V)$ occurs in $\mathcal{D}(X)$. Now, I does not satisfy any literal in this clause, since all variables in V are mapped to 1 and all variables in $\mathrm{vars}(X) \setminus V$ are mapped to 0. Thus, $I \not\models \mathcal{D}(X)$, which is a contradiction. Thus, I satisfies X. □

Example 5.3 (Direct encoding of parity constraints):
Consider the following parity constraints:

$$X_1 = \mathsf{par}(\mathsf{p}_1, \mathsf{p}_2, \mathsf{p}_3, \top) \qquad X_3 = \mathsf{par}(\mathsf{p}_1, \mathsf{p}_4)$$
$$X_2 = \mathsf{par}(\mathsf{p}_2, \mathsf{p}_4, \top) \qquad X_4 = \mathsf{par}(\mathsf{p}_3, \top)$$

Their direct encodings are given below:

$$\mathcal{D}(X_1) = \{\mathsf{or}(\mathsf{p}_1, \mathsf{p}_2, \mathsf{p}_3), \mathsf{or}(\mathsf{p}_1, \neg\mathsf{p}_2, \neg\mathsf{p}_3), \mathsf{or}(\neg\mathsf{p}_1, \mathsf{p}_2, \neg\mathsf{p}_3), \mathsf{or}(\neg\mathsf{p}_1, \neg\mathsf{p}_2, \mathsf{p}_3)\}$$
$$\mathcal{D}(X_2) = \{\mathsf{or}(\mathsf{p}_2, \mathsf{p}_4), \mathsf{or}(\neg\mathsf{p}_2, \neg\mathsf{p}_4)\}$$
$$\mathcal{D}(X_3) = \{\mathsf{or}(\neg\mathsf{p}_1, \mathsf{p}_4), \mathsf{or}(\neg\mathsf{p}_1, \mathsf{p}_4)\}$$
$$\mathcal{D}(X_4) = \{\mathsf{or}(\mathsf{p}_3)\}$$

Furthermore, we obtain the following encodings for the zero-size parity constraints:

$$\mathcal{D}(\mathsf{par}(\top)) = \{\mathsf{or}(\emptyset)\}$$
$$\mathcal{D}(\mathsf{par}(\emptyset)) = \emptyset$$ ■

As can be glimpsed in the example above, the size of direct encodings is exponential on the size of the encoded parity constraint. In particular, we have the following result:

Proposition 5.4 (Size of direct encodings):
Let X be a parity constraint.

i) If $|X| > 0$, then $|\mathcal{D}(X)| = 2^{|X|-1}$.

ii) If $X = \mathsf{par}(\top)$, then $|\mathcal{D}(X)| = 1$.

iii) If $X = \mathsf{par}(\emptyset)$, then $|\mathcal{D}(X)| = 0$.

Proof. Note that, for any two distinct sets V, V' of variables as in the definition of $\mathcal{D}(X)$, the clauses $\mathsf{nand}(V) \vee \mathsf{or}(\mathrm{vars}(X) \setminus V)$ and $\mathsf{nand}(V') \vee \mathsf{or}(\mathrm{vars}(X) \setminus V')$ are distinct. This gives us a bijection between clauses in $\mathcal{D}(X)$ and subsets V of $\mathrm{vars}(X)$ with $|V| \approx \mathrm{pol}(X) + 1$. We conclude that, if we define the set $S = \{V \subseteq \mathrm{vars}(X) : |V| \approx \mathrm{pol}(X) + 1\}$, we obtain $|\mathcal{D}(X)| = |S|$. Proposition 2.2 then gives the claim. □

This exponential behavior of the direct encoding stimulated research to obtain smaller encodings. A widespread general approach to do so is to use *fresh variables* to split the encodings [Pre09]. Fresh variables are propositional variables that are used nowhere in the original CNF formula, being therefore suitable to introduce definitions without compromising the satisfiability of the problem. Further desirable properties include *maintenance of generalized arc-consistency* [MPS14, Bes06], which guarantees that unit propagation produces all relevant assignments whenever sufficient information has been derived by the SAT solver. The matter of arc-consistent encodings lies beyond the scope of this thesis; a comprehensive study of parity constraints has been developed by Matthew Gwynne and Oliver Kullmann [GK13], who showed that no polynomially-sized encoding of affine formulae maintains generalized arc-consistency.

In their work, the *natural splitting* encoding is presented. However, this encoding depends on a case distinction that makes results in our work very lengthy to present. We introduce a modification of their natural splitting encoding without case discussion, called *regularized splitting*. Hereinafter, we will refer to regularized splitting simply by "splitting".

The use of splittings requires that fresh variables are available. To formalize this idea, we assume that the set of all variables **Var** is partitioned into two countably infinite sets of variables **BVar** and **TVar**. Variables in **BVar** are called *basic variables*, and correspond to common-use variables. A clause, CNF formula,

parity constraint or affine formula θ is said to be *basic* whenever all variables in $\mathrm{vars}(\theta)$ are basic; a string of these constraints is basic whenever all constraints along the string are basic. On the other hand, variables in **TVar** are said to be *Tseitin variables*, and correspond to fresh variables. We assume that **TVar** contains a distinct variable u_i^X for every basic parity constraint X and every $i \in \mathbb{N}$. Variables u_i^X are called *splitting variables*. For any basic parity constraint X and every basic affine formula A, we define:

$$\mathbf{SVar}(X) = \{\mathsf{u}_i^X : i \in \mathbb{N}\} \quad \mathbf{SVar}(A) = \{\mathsf{u}_i^Y : Y \in A \text{ and } i \in \mathbb{N}\} \quad \mathbf{SVar} = \{\mathsf{u}_i^Y : Y \in \mathbf{Par} \text{ and } i \in \mathbb{N}\}$$

We further assume a total ordering $<$ over basic variables. In our examples, we assume that variables denoted by p_i are basic, and $\mathsf{p}_i < \mathsf{p}_j$ whenever $i < j$.

Definition 5.5 (Matrix, regularized splitting and linear encoding):
Let X be a basic parity constraint and $\{a_1, \ldots, a_n\}$ an enumeration of $\mathrm{vars}(X)$ *with $a_1 < a_2 < \cdots < a_n$. The* matrix *of X is defined as the affine formula:*

$$\mathcal{M}(X) = \{\mathsf{par}(\mathsf{u}_0^X)\} \cup \{\mathsf{par}(\mathsf{u}_{i-1}^X, a_i, \mathsf{u}_i^X) : 1 \leq i \leq n\}$$

We define the (regularized) splitting *of X as the affine formula $\mathcal{S}(X) = \mathcal{M}(X) \cup \{\mathsf{par}(\mathsf{u}_n^X)_{(X)}\}$. The additional parity constraint $\mathsf{par}(\mathsf{u}_n^X)_{(X)}$ is called the* independent parity constraint *of X. For a basic affine formula A, we define its regularized splitting as the affine formula $\mathcal{S}(A) = \bigcup_{X \in A} \mathcal{S}(X)$.*

The linear encoding $\mathcal{L}(X)$ *of a basic parity constraint X is defined as the CNF formula $\mathcal{D}(\mathcal{S}(X))$, i.e. the direct translation of its splitting. Similarly, we define for a basic affine formula A its linear encoding as $\mathcal{L}(A) = \mathcal{D}(\mathcal{S}(A))$.*

Example 5.6 (Splitting and linear encoding of parity constraints):
Consider parity constraints $X = \mathsf{par}(\mathsf{p}_1, \mathsf{p}_5, \top)$ and $Y = \mathsf{par}(\mathsf{p}_0, \mathsf{p}_2, \mathsf{p}_4)$, with splittings given by:

$$\mathcal{S}(X) = \{\mathsf{par}(\mathsf{u}_0^X), \mathsf{par}(\mathsf{u}_0^X, \mathsf{p}_1, \mathsf{u}_1^X), \mathsf{par}(\mathsf{u}_1^X, \mathsf{p}_5, \mathsf{u}_2^X), \mathsf{par}(\mathsf{u}_2^X, \top)\}$$
$$\mathcal{S}(Y) = \{\mathsf{par}(\mathsf{u}_0^Y), \mathsf{par}(\mathsf{u}_0^Y, \mathsf{p}_0, \mathsf{u}_1^Y), \mathsf{par}(\mathsf{u}_1^Y, \mathsf{p}_2, \mathsf{u}_2^Y), \mathsf{par}(\mathsf{u}_2^Y, \mathsf{p}_4, \mathsf{u}_3^Y), \mathsf{par}(\mathsf{u}_3^Y)\}$$

The independent parity constraint of each one is the rightmost parity constraint in the splittings above, whereas their matrices contain all other parity constraints in each splitting. Note that $\top$ only occurs in the independent parity constraint if it occurs in the original parity constraint. The linear encoding $\mathcal{L}(X)$ of the parity constraint X is the following affine formula:

$$\Big\{ \mathsf{or}(\neg\mathsf{u}_0^X)\ ,\ \mathsf{or}(\neg\mathsf{u}_0^X, \neg\mathsf{p}_1, \neg\mathsf{u}_1^X)\ ,\ \mathsf{or}(\neg\mathsf{u}_0^X, \mathsf{p}_1, \mathsf{u}_1^X)\ ,\ \mathsf{or}(\mathsf{u}_0^X, \neg\mathsf{p}_1, \mathsf{u}_1^X)\ ,\ \mathsf{or}(\mathsf{u}_0^X, \mathsf{p}_1, \neg\mathsf{u}_1^X)$$
$$\mathsf{or}(\neg\mathsf{u}_1^X, \neg\mathsf{p}_5, \neg\mathsf{u}_2^X)\ ,\ \mathsf{or}(\neg\mathsf{u}_1^X, \mathsf{p}_5, \mathsf{u}_2^X)\ ,\ \mathsf{or}(\mathsf{u}_1^X, \neg\mathsf{p}_5, \mathsf{u}_2^X)\ ,\ \mathsf{or}(\mathsf{u}_1^X, \mathsf{p}_5, \neg\mathsf{u}_2^X)\ ,\ \mathsf{or}(\mathsf{u}_2^X) \Big\} \qquad \blacksquare$$

From Proposition 5.2 we know that the splitting of an affine formula and its linear encoding are semantically equivalent. However, they are not semantically equivalent to the original affine formula. We show that this is only due to the use of splitting variables: every interpretation satisfying the original affine formula can be turned into an interpretation satisfying the splitting by changing its values over the splitting variables.

Theorem 5.7:
[$\mathcal{L}(X)$ is a CNF encoding of X] Let X be a basic parity constraint and I an interpretation. The following hold:

i) I satisfies $\mathcal{S}(X)$ if and only if $\mathcal{L}(X)$.

ii) If I satisfies $\mathcal{S}(X)$, then I satisfies X.

iii) If I satisfies X, then there is a subset $V \subseteq \mathbf{SVar}(X)$ such that I/V satisfies $\mathcal{S}(X)$.

Proof.

i) Follows straightforward from Proposition 5.2 and considering that $\mathcal{L}(X) = \mathcal{D}(\mathcal{S}(X))$.

ii) Let $\{a_1, \ldots, a_n\}$ be an enumeration of $\mathrm{vars}(X)$ with $a_1 < \cdots < a_n$. We show by induction that I satisfies the parity constraint $Y_i = \mathsf{par}(a_1, \ldots, a_i, \mathsf{u}_i^X)$ for $0 \leq i \leq n$ by induction on i. For $i = 0$ we have $Y_0 = \mathsf{par}(\mathsf{u}_0^X)$ which is included in $\mathcal{S}(X)$, so it is satisfied by I.

Assume now that I satisfies the parity constraint $Y_i = \mathsf{par}(a_1, \ldots, a_i, \mathsf{u}_i^X)$ for an $i < n$. The parity constraint $\mathsf{par}(\mathsf{u}_i^X, a_{i+1}, \mathsf{u}_{i+1}^X)$ occurs in $\mathcal{S}(X)$, hence it is satisfied by I. Now observe that:

$$Y_{i+1} = \mathsf{par}(a_1, \ldots, a_{i+1}, \mathsf{u}_{i+1}^X) = Y_i \oplus \mathsf{par}(\mathsf{u}_i^X, a_{i+1}, \mathsf{u}_{i+1}^X)$$

By Proposition 2.21, we conclude that I satisfies Y_{i+1}.

Thus we conclude by induction that I satisfies the parity constraint $Y_n = \mathsf{par}(a_1, \ldots, a_n, \mathsf{u}_n^X)$. Note that $\mathrm{pol}(Y_n) = 0$. Therefore, the polarity of the parity constraint $Z = Y_n \oplus \mathsf{par}(\mathsf{u}_n^X)_{(X)}$ is that of X. In particular, we obtain that $\mathrm{vars}(Z) = \{a_1, \ldots, a_n\}$ and $\mathrm{pol}(Z) = \mathrm{pol}(X)$, which means that $Z = X$. Now, I satisfies both Y_n and $\mathsf{par}(\mathsf{u}_n^X)_{(X)}$, so it satisfies X as well.

iii) As above, let $\{a_1, \ldots, a_n\}$ be an enumeration of $\mathrm{vars}(X)$ with $a_1 < \cdots < a_n$. Define the set

$$V = \left\{ \mathsf{u}_i^X : I(\mathsf{u}_i^X) \not\approx \sum_{j=1}^{i} I(a_j) \right\}$$

and let J be the interpretation I/V. Observe that $\sum_{j=1}^{0} I(a_j) = 0$, which means that u_0^X is in V if and only if $I(\mathsf{u}_0^X) = 1$. Hence, $J(\mathsf{u}_0^X) = 0$, and thus J satisfies the parity constraint $\mathsf{par}(\mathsf{u}_0^X)$.

Let i be an integer with $0 \leq i \leq n$. If $I(\mathsf{u}_i^X) \approx \sum_{j=1}^{i} I(a_j)$, then i does not belong to V, and so $J(\mathsf{u}_i^X) = I(\mathsf{u}_i^X)$; otherwise, $i \in V$ and then $J(\mathsf{u}_i^X) \not\approx I(\mathsf{u}_i^X)$. In both cases we can conclude that $J(\mathsf{u}_i^X) \approx \sum_{j=1}^{i} I(a_j)$. Then, we have that, for $1 \leq i \leq n$,

$$J(\mathsf{u}_{i-1}^X) + J(a_i) + J(\mathsf{u}_i^X) \approx \sum_{j=1}^{i-1} I(a_j) + I(a_i) + \sum_{j=1}^{i} I(a_j) \approx 0$$

which shows that J satisfies the parity constraint $\mathsf{par}(\mathsf{u}_{i-1}^X, a_i, \mathsf{u}_i^X)$. Finally, since the a_i variables do not occur in V, if I satisfies X then so does J. This implies that $\sum_{i=1}^{n} J(a_i) \approx 1$ if and only if $\top \in \mathrm{vars}^\star(X)$, or equivalently, if $\top \in \mathsf{par}(\mathsf{u}_n^X)_{(X)}$. Since $J(\mathsf{u}_b^X) \approx \sum_{j=1}^{n} I(a_j)$, this implies that J satisfies $\mathsf{par}(\mathsf{u}_n^X)_{(X)}$, and hence $J \models \mathcal{S}(X)$. □

The main advantage of the linear encoding is that its size is linear on the size of the original parity constraint, as shown in Proposition 5.8. In this Thesis, this will be essential to contain the exponential blow up in translations due to the size of intermediate steps.

Proposition 5.8 (Size of linear encodings):
Let X be a parity constraint. Then,

i) $\mathcal{S}(X)$ *contains two parity constraints of size* 1 *and* $|X|$ *parity constraints of size* 3.

ii) $|\mathcal{L}(X)| = 2 + 4|X|$.

Proof.

i) Straightforward from the definition of the splitting.

ii) Size-1 parity constraints in $\mathcal{S}(X)$ are transformed by $\mathcal{D}$ into $2^{1-1} = 1$ clauses; size-3 parity constraints are transformed into $2^{3-1} = 4$ clauses. Now, there are 2 parity constraints of size 1 and $|X|$ parity constraints of size 3 in $\mathcal{S}(X)$, which totals to $2 + 4|X|$ clauses in $\mathcal{L}(X)$. □

5.2 Parity reasoning

Before we can show how parity reasoning procedures are integrated in SAT solvers, a description of the available parity reasoning procedures is necessary. Current SAT solvers with integrated parity reasoning use a variety of reasoning procedures, such as *equivalence reasoning* [HvM05, LJN11, Li03], *Gaussian elimination* [SNC09, Soo12] or *Gauss-Jordan elimination* [LJN12, HJ12], *parity constraint cutting* [SNC09], *dependent variable elimination* [WvM98], *global substitution* [Heu14], *row and column elimination* [Soo12], and *block decomposition* [Soo12]. To the best of our knowledge, the underlying proof system to all these procedures is the **EXor** proof system. We present here the Gauss-Jordan elimination procedure, which is the most powerful [LJN12, HJ12].

Gauss-Jordan elimination yields an affine formula semantically equivalent to the input, with the property that every parity constraint is a XOR definition in the rest of the affine formula. Such an affine system has interesting properties for SAT solving, because unsatisfiability (corresponding to conflict detection in SAT solving) and entailed literals (corresponding to unit propagation) are explicit from the output.

Definition 5.9 (Reduced affine formulae):
An affine formula A is said to be reduced *whenever one of the following hold:*

a) $A = \{\mathsf{par}(\top)\}$.

b) Every parity constraint $X \in A$ is a XOR definition in $A \setminus \{X\}$.

Reduced affine formulae correspond to reduced row echelon form matrices for an adequate choice of columns and rows, as shown in the following example.

Example 5.10 (Reduction of affine formulae):
Recall the affine formula A given in Example 2.24, which was transformed by addition steps into the semantically equivalent affine formula B.

$$A = \{\mathsf{par}(\mathsf{p}_1, \mathsf{p}_2, \mathsf{p}_3, \top)\,,\ \mathsf{par}(\mathsf{p}_2, \mathsf{p}_4, \top)\,,\ \mathsf{par}(\mathsf{p}_1, \mathsf{p}_4)\}$$
$$B = \{\mathsf{par}(\mathsf{p}_2, \mathsf{p}_4, \top)\,,\ \mathsf{par}(\mathsf{p}_1, \mathsf{p}_4)\,,\ \mathsf{par}(\mathsf{p}_3)\}$$

A is not a reduced affine formula, since all variables occurring in $\mathsf{par}(\mathsf{p}_2, \mathsf{p}_4, \top)$ occur in some other parity constraint in A. The affine formula B is reduced, because the variable p_2 only occurs in the first parity constraint, p_1 only occurs in the second and p_3 only occurs in the third. Furthermore, it is not trivial from the definition of A that $\neg\mathsf{p}_3$ is entailed. However, this is straightforward from the equivalent affine formula B, since $\mathrm{xor}(\neg\mathsf{p}_3) = \mathsf{par}(\mathsf{p}_3) \in B$. ■

Proposition 5.11 (Properties of reduced affine formulae):
Let A be a reduced affine formula. The following hold:

i) A is unsatisfiable if and only if $A = \{\mathsf{par}(\top)\}$.

ii) If A is satisfiable, then $A \models l$ if and only if $\mathrm{xor}(l) \in A$.

iii) If A is satisfiable, then $|A| \leq |\mathrm{vars}(A)|$; otherwise $|A| = 1$.

Proof.

i) The "if" implication is straightforward. We show the "only if" implication by proving that, whenever $A \neq \{\mathsf{par}(\top)\}$ holds, then A is satisfiable. By Definition 5.9, the hypothesis implies that every parity constraint $X \in A$ is a XOR definition in $A \setminus \{X\}$. Therefore, there must be a variable $a_X \in \mathrm{vars}(X)$ for every $X \in A$ such that $a_X \notin \mathrm{vars}(A \setminus \{X\})$. Let I be any interpretation, and define the interpretation

$J = I/V$ for the set of variables:

$$V = \{a_X : X \in A \text{ and } I \not\models X\}$$

For every parity constraint $X \in A$ we find that, for all variables b in $\text{vars}(X)$ other than a_X, then I and J coincide over b. Observe that $|V \cap \text{vars}(X)|$ is odd if I falsifies X, and even otherwise. By Proposition 2.21 we obtain that J satisfies X, for every parity constraint $X \in A$. Thus, A is satisfiable.

ii) Again, the "if" implication is straightforward. To show the "only if" implication, observe that A does not contain the unsatisfiable parity constraint $\mathsf{par}(\top)$, because A is satisfiable. This implies, by Definition 5.9, that for every constraint $X \in A$ there is some variable $a_X \in \text{vars}(X)$ such that $a_X \notin \text{vars}(A \setminus \{X\})$. Define the set:

$$V = \{b \in \text{vars}(A) : b \neq a_X \text{ for all } X \in A\}$$

We first show the following claim

> For every Boolean interpretation I there is another interpretation I' satisfying A with $I(b) = I'(b)$ for all $b \in V$.

To show this, define $U = \{a_X : X \in A \text{ and } I \not\models X\}$, and consider the Boolean interpretation $I' = I/U$. For every parity constraint $X \in A$, there is one variable in $\text{vars}(X) \cap U$ if I falsifies X, and no variables otherwise. By Proposition 2.21, I' satisfies A. Furthermore, I and $I' = I/U$ differ only in variables that do not occur in V, so every variable $b \in V$ is mapped to the same value by I and I', and the claim above follows.

Now we show the Proposition. The claim above implies that $\text{var}(l)$ must be one of the a_X variables; otherwise, we would be able to find interpretations I, I' satisfying A with $I(\text{var}(l)) \neq I'(\text{var}(l))$, which would contradict $A \models l$. Let X be the (unique) parity constraint in A with $\text{var}(l) = a_X$. Since A is satisfiable, let us consider an interpretation I satisfying A. If X contained an occurrence of a variable b other than a_X, then we can consider the interpretation $J = I/\{b\}$, and by the claim above, derive the existence of an interpretation J' satisfying A such that $J(a) = J'(a)$ for every $a \in V$. Note that this implies that all variables $a \in \text{vars}(X) \setminus \{\text{var}(l), b\}$ are mapped to the same value by all I, J and J'. Furthermore, we have $I(b) \neq J(b) = J'(b)$. Since J' satisfies X, this can only happen if $I(\text{var}(l)) \neq I'(\text{var}(l))$, and we thus conclude that $A \models l$, which is a contradiction rising from the assumption that X contains variables other than $a_X = \text{var}(l)$.

Therefore, $\text{vars}(X) = \{\text{var}(l)\}$, which implies that either $X = \mathsf{par}(\text{var}(l))$ or $X = \mathsf{par}(\text{var}(l), \top)$ holds. From the fact that $A \models l$, this implies that $X \models l$, and hence it is only possible that $X = \text{xor}(l)$, which finishes the proof.

iii) Follows straightforward from Definition 5.9. □

The Gauss-Jordan elimination algorithm shown in Algorithm 1 transforms an arbitrary affine formula A into a semantically equivalent reduced affine formula A'. To achieve this, the algorithm starts with a copy A' of A. A non-trivial parity constraint X in A' that is not yet a XOR definition in $A' \setminus \{X\}$ and a variable a occurring in X are (don't-care) non-deterministically selected. Then, a is removed from all other constraints in A' by adding X to all those parity constraints containing a. When this process finishes, A' only contains parity constraints that either contain no variables, or are XOR definitions in the rest of the affine formula. The output A' is $\{\mathsf{par}(\top)\}$ if the unsatisfiable constraint occurs in A'; otherwise, occurrences of $\mathsf{par}(\emptyset)$ in A' are removed before returning it. In the pseudocode presented as Algorithm 1, an **Xor**-derivation of A' from A is generated as well.

Input: an affine formula A.
Output: a reduced affine formula A' with $A \equiv A'$.
Output: an **Xor**-derivation φ of A' from A.

1 $A' \coloneqq A$
2 $\varphi \coloneqq [A']$
3 **while** there is an $X \in A'$ with $\emptyset \neq \mathrm{vars}(X) \subseteq \mathrm{vars}(A' \setminus \{X\})$ **do**
4 **pick** $X \in A'$ **such that** $\emptyset \neq \mathrm{vars}(X) \subseteq \mathrm{vars}(A' \setminus \{X\})$
5 **pick** $a \in \mathrm{vars}(X)$
6 **while** there is an $Y \in A' \setminus \{X\}$ with $a \in \mathrm{vars}(Y)$ **do**
7 **pick** $Y \in A' \setminus \{X\}$ **such that** $a \in \mathrm{vars}(Y)$
8 $\varphi \coloneqq \varphi : [A' \cup \{X \oplus Y\}\ ,\ A' \setminus \{Y\} \cup \{X \oplus Y\}]$
9 $A' \coloneqq A' \setminus \{Y\} \cup \{X \oplus Y\}$
10 **if** $\mathsf{par}(\top) \in A'$ **then**
11 $A' \coloneqq \{\mathsf{par}(\top)\}$
12 **else**
13 $A' \coloneqq A' \setminus \{\mathsf{par}(\emptyset)\}$
14 $\varphi \coloneqq \varphi : [A']$

Algorithm 1: Gauss-Jordan elimination algorithm.

Theorem 5.12 (Correctness of Gauss-Jordan elimination):
Let A be an affine formula. Apply Algorithm 1 with input affine formula A to obtain the affine formula A' and the string φ of affine formulae. The following hold:

i) Algorithm 1 terminates.

ii) A' is a reduced affine formula with $A \equiv A'$.

iii) φ is an **Xor***-derivation of A' from A.*

Proof.

i) It suffices to show that the while-loops starting in lines 3 and 6 terminate. For the latter, observe that a parity constraint $X \in A'$ and a variable $a \in \mathrm{vars}(X)$ are fixed. In every iteration through lines 6–9 the algorithm replaces a parity constraint $Y \in A'$ other than X containing a by $X \oplus Y$. Since a occurs in both X and Y, it does not occur in $X \oplus Y$. Therefore, the number of constraints in $A' \setminus \{X\}$ is reduced by 1 in every iteration through this while-loop, and it is exited when this number is 0. Hence, the while-loop starting in line 6 terminates every time it is entered.

For the while-loop starting in line 3, a parity constraint X is selected in A' sharing all its variables with other constraints in A'. The algorithm then picks a variable a in X and, by the end of the while-loop starting in line 6, X is the only parity constraint in A' containing a. The number of parity constraints sharing all their variables with other constraints in A' is then reduced in every iteration through the while-loop starting in line 3, and this while-loop is exited when that number reaches 0. Therefore, this while-loop terminates as well, and the algorithm terminates.

ii) For the reasons aforementioned, by the time the algorithm reaches line 10 A' does not contain parity constraints that share all their variables with other parity constraints in A'. This means that by line 10 A' only may contain the empty parity constraint $\mathsf{par}(\emptyset)$, the unsatisfiable parity constraint $\mathsf{par}(\top)$ or parity constraints X that are XOR definitions in $A' \setminus \{X\}$. Observe that this is achieved by replacing parity constraints Y by their addition $X \oplus Y$ with a different parity constraint X from A. Now, since

$Y = (X \oplus Y) \oplus Y$, doing so preserves semantic equivalence, and so A' in line 10 is semantically equivalent to A.

If A' contains the unsatisfiable parity constraint $\mathsf{par}(\top)$, then it is unsatisfiable, and $\{\mathsf{par}(\top)\}$ is returned as A', which is reduced and semantically equivalent to A. Otherwise, A' only contains the empty parity constraint $\mathsf{par}(\emptyset)$ and XOR definitions X in $A' \setminus \{X\}$. All of the latter are XOR definitions in $A' \setminus \{X, \mathsf{par}(\top)\}$ as well, and A' is semantically equivalent to $A' \setminus \{\mathsf{par}(\top)\}$, which justifies that the returned A' is reduced and equivalent to A.

iii) It follows from considering that in lines 9 and 14 the last affine formula replicates A'. Furthermore, every modification in these lines is a correct **Xor** inference. □

Observe that this algorithm can be executed in polynomial time on the size of A, and furthermore the output **Xor**-derivation has quadratic size on $|A|$. In general, in this Thesis we assume that **Xor**-derivations to be translated are quadratically-sized on the size of the premise affine formula.

5.3 Gauss-Jordan elimination for SAT solving

As explained in the introduction of this chapter, separation results in proof complexity theory [Urq87] show that the resolution proof system underlying classical CDCL-style SAT solvers [PD11] can only exponentially long proofs when parity constraints [GK13] and pseudo-Boolean constraints [RM09] are encoded in the input CNF formula. However, other proof systems obtain polynomial proofs in these fragments, such as the XOR proof system or the *cutting-planes* proof system [Hoo88]. Since the underlying proof system of any solving procedure poses a lower bound to time complexity of proving procedures the idea of developing solvers that integrate CDCL procedures with specialized solvers for parity or cardinality reasoning has been explored in the last decade.

For example, *cardinality* or *pseudo-Boolean constraints* can be detected in the CNF formula [BBLM14], and then reasoning procedures such as *Fourier-Motzkin elimination* [Mot36] can be used to simplify them [SS06, Man14b]. The underlying proof system to Fourier-Motzkin elimination is the *cutting-planes* proof system, which is strictly more powerful than resolution from a proof complexity perspective [Hoo88, CCT87]. Similarly, Gauss-Jordan elimination modulo 2 yields polynomially-sized XOR proofs for affine formulae whereas the results by Urquhart [Urq87] prove this impossible for their direct CNF encodings with resolution. This is especially troublesome for problems in cryptography [MM00, CB07], where parity constraints arise naturally, but not restricted to them: `coProcessor` [Man11] is able to detect non-trivial parity constraints in 70% of the instances in the application track from SAT Competition 2014. Solvers incorporating parity reasoning modules based on Gauss-Jordan elimination or restrictions thereof include `Riss` [Man14a], `cryptoMiniSat` [Soo14], `eqSatz` [Li00] and `march_eq` [HDvZvM04].

Two mutually compatible methods for applying Gauss-Jordan elimination in CDCL-style solvers have been proposed; we refer to them as inprocessing and interleaved parity reasoning. *Inprocessing* parity reasoning detects parity constraints encoded in the CNF formulae and performs Gauss-Jordan elimination as an inprocessing step to simplify them. On the other hand, *interleaved* parity reasoning integrates Gauss-Jordan elimination into the CDCL procedure by allowing propagation of literals entailed by parity constraints and detecting conflicts directly in the affine formula.

5.3.1 Inprocessing parity reasoning

Inprocessing parity reasoning is a technique that performs Gauss-Jordan elimination over an affine formula encoded in the CNF formula as an inprocessing step, i.e. between cycles of the CDCL procedure. The work of Mate Soos, Karsten Nohl and Claude Castelluccia [SNC09, Soo12] allowed a successful application of

$\langle F \cup \mathcal{D}(A), [] \rangle \xrightarrow[\text{X-simp}]{} \langle F \cup \mathcal{D}(A'), [] \rangle$	iff	A and A' are affine formulae with $A \equiv A'$.
$\langle F \cup \mathcal{D}(A), \nu \rangle \xrightarrow[\text{X-unit}]{} \langle F \cup \mathcal{D}(A), \nu : [l] \rangle$	iff	$A \cup \text{xor}(\nu) \models l$, the affine formula $A \cup \text{xor}(\nu)$ is satisfiable and $l \notin \text{lits}(\nu)$.
$\langle F \cup \mathcal{D}(A), \nu \rangle \xrightarrow[\text{X-learn}]{} \langle F \cup \mathcal{D}(A) \cup \{\mathcal{C}(\sigma)\}, \nu \rangle$	iff	K is an implication graph XOR-compatible with $\langle F \cup \mathcal{D}(A), \nu \rangle$ and σ is a shift on K.

Table 5.2: Rules for XOR-CDCL SAT solving.

SAT solvers to cryptographic problems. This is a widespread, general-use technique, either in its *equivalence reasoning* form [WvM98, Heu08] or in its complete form with *Gauss-Jordan elimination* [Man14b]. The inprocessing parity reasoning method consists of two steps. In the first step, direct encodings of parity constraints are detected in the CNF formula. An efficient algorithm to do so is proposed in [HvM05]. In this step, given a CNF formula F, an affine formula A is found such that $\mathcal{D}(A) \subseteq F$. In the second step, Gauss-Jordan elimination is applied to A, which yields the semantically equivalent reduced affine formula A'. Then, the CNF formula F is simplified to the equivalent CNF formula $F \setminus \mathcal{D}(A) \cup \mathcal{D}(A')$. Inprocessing parity reasoning is modeled by the rule $\xrightarrow[\text{X-simp}]{}$ in Table 5.2.

Example 5.13 (Inprocessing parity reasoning):
Consider the CNF formula F containing the clauses listed below:

$$
\begin{array}{llllll}
C_1 = & \mathsf{or}(\neg p_1, p_2, p_3, p_4) & C_2 = & \mathsf{or}(p_1, \neg p_2, p_3, p_4) & C_3 = & \mathsf{or}(p_1, p_2, \neg p_3, p_4) \\
C_4 = & \mathsf{or}(p_1, p_2, p_3, \neg p_4) & C_5 = & \mathsf{or}(\neg p_1, \neg p_2, \neg p_3, p_4) & C_6 = & \mathsf{or}(\neg p_1, \neg p_2, p_3, \neg p_4) \\
C_7 = & \mathsf{or}(\neg p_1, p_2, \neg p_3, \neg p_4) & C_8 = & \mathsf{or}(p_1, \neg p_2, \neg p_3, \neg p_4) & C_9 = & \mathsf{or}(p_1, p_2, p_5, p_6) \\
C_{10} = & \mathsf{or}(\neg p_1, \neg p_2, p_5, p_6) & C_{11} = & \mathsf{or}(\neg p_1, p_2, \neg p_5, p_6) & C_{12} = & \mathsf{or}(\neg p_1, p_2, p_5, \neg p_6) \\
C_{13} = & \mathsf{or}(p_1, \neg p_2, \neg p_5, p_6) & C_{14} = & \mathsf{or}(p_1, \neg p_2, p_5, \neg p_6) & C_{15} = & \mathsf{or}(p_1, p_2, \neg p_5, \neg p_6) \\
C_{16} = & \mathsf{or}(\neg p_1, \neg p_2, \neg p_5, \neg p_6) & C_{17} = & \mathsf{or}(p_3, p_4, p_5, p_6) & C_{18} = & \mathsf{or}(\neg p_3, \neg p_4, p_5, p_6) \\
C_{19} = & \mathsf{or}(\neg p_3, p_4, \neg p_5, p_6) & C_{20} = & \mathsf{or}(\neg p_3, p_4, p_5, \neg p_6) & C_{21} = & \mathsf{or}(p_3, \neg p_4, \neg p_5, p_6) \\
C_{22} = & \mathsf{or}(p_3, \neg p_4, p_5, \neg p_6) & C_{23} = & \mathsf{or}(p_3, p_4, \neg p_5, \neg p_6) & C_{24} = & \mathsf{or}(\neg p_3, \neg p_4, \neg p_5, \neg p_6) \\
C_{25} = & \mathsf{or}(\neg p_2, p_3, p_5) & C_{26} = & \mathsf{or}(p_2, \neg p_3, p_5) & C_{27} = & \mathsf{or}(p_2, p_3, \neg p_5) \\
C_{28} = & \mathsf{or}(\neg p_2, \neg p_3, \neg p_5) & C_{29} = & \mathsf{or}(p_1, \neg p_8) & C_{30} = & \mathsf{or}(p_2, \neg p_8) \\
C_{31} = & \mathsf{or}(p_3, p_{13}) & C_{32} = & \mathsf{or}(\neg p_3, p_7) & C_{33} = & \mathsf{or}(p_6, \neg p_{12}) \\
C_{34} = & \mathsf{or}(p_6, \neg p_{10}, \neg p_{11}) & C_{35} = & \mathsf{or}(p_6, p_{11}, p_{12}, \neg p_{13}) & C_{36} = & \mathsf{or}(p_6, \neg p_7, p_8) \\
C_{37} = & \mathsf{or}(p_7, p_8) & C_{38} = & \mathsf{or}(p_8, \neg p_9, p_{10}) & C_{39} = & \mathsf{or}(\neg p_8, p_{10})
\end{array}
$$

We can detect the encoding of 4 parity constraints in F, namely:

$$
\begin{array}{ll}
X_1 = \mathsf{par}(p_1, p_2, p_3, p_4) & \mathcal{D}(X_1) = \{C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_8\} \\
X_2 = \mathsf{par}(p_1, p_2, p_5, p_6, \top) & \mathcal{D}(X_2) = \{C_9, C_{10}, C_{11}, C_{12}, C_{13}, C_{14}, C_{15}, C_{16}\} \\
X_3 = \mathsf{par}(p_3, p_4, p_5, p_6, \top) & \mathcal{D}(X_3) = \{C_{17}, C_{18}, C_{19}, C_{20}, C_{21}, C_{22}, C_{23}, C_{24}\} \\
X_4 = \mathsf{par}(p_2, p_3, p_5) & \mathcal{D}(X_4) = \{C_{25}, C_{26}, C_{27}, C_{28}\}
\end{array}
$$

Now Gauss-Jordan elimination may be applied to the affine formula $A = \{X_1, X_2, X_3, X_4\}$, which yields the equivalent reduced affine formula B containing the following parity constraints:

$$
Y_1 = \mathsf{par}(p_1, p_4, p_5) \qquad Y_2 = \mathsf{par}(p_2, p_4, p_6, \top) \qquad Y_3 = \mathsf{par}(p_3, p_4, p_5, p_6, \top)
$$

The affine formula B is encoded back into CNF and the encoding $\mathcal{D}(A)$ is replaced by $\mathcal{D}(B)$. This inprocessing step is modeled as $\langle F, [] \rangle \xrightarrow[\text{X-simp}]{} \langle F', [] \rangle$, where $F' = \mathcal{D}(B) \cup \{C_{29}, \ldots, C_{39}\}$. ■

5.3.2 Interleaved parity reasoning.

Interleaved parity reasoning integrates Gauss-Jordan elimination as a reasoning engine for propagation and conflict detection inside the CDCL procedure. In last years there have been several approaches for interleaved parity reasoning. The origins of interleaved parity reasoning backtrack to Peter Baumgartner and Fabio Massacci [BM00]. Tero Laitinen, Tommi Junttila and Ilkka Niemelä [Lai14, LJN11, LJN10, LJN12, LJN14b] further developed the field with successive approaches to integrate increasingly expressive techniques for parity reasoning into SAT solvers.

Interleaved parity reasoning involves two different but related techniques. The first one is propagation of literals entailed through parity reasoning. Assume the solver is currently in a state $\langle F, \nu \rangle$, and an affine formula A is detected in the CNF formula F directly encoded, that is, $\mathcal{D}(A) \subseteq F$. There might be a literal l entailed by the CNF formula F together with the assumptions given by the partial assignment ν, i.e. a literal l with $F \cup \text{lits}(\nu) \models l$. In general, it is not easy to detect if this is the case. However, Proposition 5.11 shows that Gauss-Jordan elimination can detect if l is entailed by the affine formula together with the assumptions in ν, i.e. whether $A \cup \text{xor}(\nu) \models l$. In this case, the literal l can be added to the partial assignment ν. This technique is modeled by the rule $\xrightarrow[\text{X-unit}]{}$.

Example 5.14 (Interleaved parity reasoning: propagation):
Let us continue Example 5.13 by applying interleaved parity reasoning through the $\xrightarrow[\text{X-unit}]{}$ rule:

Rule	Assignment ν	Reduct $F\vert_\nu$
	$[]$	F
$\xrightarrow[\text{X-simp}]{}$	$[]$	$F' = \{C_{29}, \ldots, C_{39}\} \cup \mathcal{D}(B)$
$\xrightarrow[\text{decide}]{}$	$[\neg p_7^\bullet]$	$\{\text{or}(p_1, \neg p_8), \text{or}(p_2, \neg p_8), \text{or}(p_3, p_{13}), \text{or}(\neg p_3), \text{or}(p_6, \neg p_{12}), \text{or}(p_6, \neg p_{10}, \neg p_{11}), \text{or}(p_6, p_{11}, p_{12}, \neg p_{13}), \text{or}(p_8), \text{or}(p_8, \neg p_9, p_{10}), \text{or}(\neg p_8, p_{10})\} \cup \mathcal{D}(B)\vert_\nu$
$\xrightarrow[\text{unit}]{}$	$[\neg p_7^\bullet, \neg p_3]$	$\{\text{or}(p_1, \neg p_8), \text{or}(p_2, \neg p_8), \text{or}(p_{13}), \text{or}(p_6, \neg p_{12}), \text{or}(p_6, \neg p_{10}, \neg p_{11}), \text{or}(p_6, p_{11}, p_{12}, \neg p_{13}), \text{or}(p_8), \text{or}(p_8, \neg p_9, p_{10}), \text{or}(\neg p_8, p_{10})\} \cup \mathcal{D}(B)\vert_\nu$
$\xrightarrow[\text{unit}]{}$	$[\neg p_7^\bullet, \neg p_3, p_8]$	$\{\text{or}(p_1), \text{or}(p_2), \text{or}(p_{13}), \text{or}(p_6, \neg p_{12}), \text{or}(p_6, \neg p_{10}, \neg p_{11}), \text{or}(p_6, p_{11}, p_{12}, \neg p_{13}), \text{or}(p_{10})\} \cup \mathcal{D}(B)\vert_\nu$
$\xrightarrow[\text{unit}]{}$	$[\neg p_7^\bullet, \neg p_3, p_8, p_1]$	$\{\text{or}(p_2), \text{or}(p_{13}), \text{or}(p_6, \neg p_{12}), \text{or}(p_6, \neg p_{10}, \neg p_{11}), \text{or}(p_6, p_{11}, p_{12}, \neg p_{13}), \text{or}(p_{10})\} \cup \mathcal{D}(B)\vert_\nu$
$\xrightarrow[\text{unit}]{}$	$[\neg p_7^\bullet, \neg p_3, p_8, p_1, p_2]$	$\{\text{or}(p_{13}), \text{or}(p_6, \neg p_{12}), \text{or}(p_6, \neg p_{10}, \neg p_{11}), \text{or}(p_6, p_{11}, p_{12}, \neg p_{13}), \text{or}(p_{10})\} \cup \mathcal{D}(B)\vert_\nu$
$\xrightarrow[\text{unit}]{}$	$[\neg p_7^\bullet, \neg p_3, p_8, p_1, p_2, p_{13}]$	$\{\text{or}(p_6, \neg p_{12}), \text{or}(p_6, \neg p_{10}, \neg p_{11}), \text{or}(p_6, p_{11}, p_{12}), \text{or}(p_{10})\} \cup \mathcal{D}(B)\vert_\nu$
$\xrightarrow[\text{unit}]{}$	$[\neg p_7^\bullet, \neg p_3, p_8, p_1, p_2, p_{13}, p_{10}]$	$\{\text{or}(p_6, \neg p_{12}), \text{or}(p_6, \neg p_{11}), \text{or}(p_6, p_{11}, p_{12})\} \cup \mathcal{D}(B)\vert_\nu$

Unit propagation is no longer possible, even in the encoded affine formula $\mathcal{D}(B)$. Classical CDCL-style solvers would now need to guess some literals before continuing. However, in the XOR-CDCL approach, propagation is still possible via the $\xrightarrow[\text{X-unit}]{}$ rule. Consider the affine formula $B \cup \{\text{xor}(p_1), \text{xor}(p_2), \text{xor}(\neg p_3)\}$

containing the following parity constraints:

$$Y_1 = \mathsf{par}(\mathsf{p}_1, \mathsf{p}_4, \mathsf{p}_5) \qquad Y_2 = \mathsf{par}(\mathsf{p}_2, \mathsf{p}_4, \mathsf{p}_6, \top) \qquad Y_3 = \mathsf{par}(\mathsf{p}_3, \mathsf{p}_4, \mathsf{p}_5, \mathsf{p}_6, \top)$$
$$Z_1 = \mathrm{xor}(\mathsf{p}_1) = \mathsf{par}(\mathsf{p}_1, \top) \qquad Z_2 = \mathrm{xor}(\mathsf{p}_2) = \mathsf{par}(\mathsf{p}_2, \top) \qquad Z_3 = \mathrm{xor}(\neg \mathsf{p}_3) = \mathsf{par}(\mathsf{p}_3)$$

We apply Gauss-Jordan elimination to this affine formula through the following steps:

Input parity constraints:

$$\begin{aligned} Y_1 &= \mathsf{par}(\mathsf{p}_1, \mathsf{p}_4, \mathsf{p}_5) & Z_1 &= \mathsf{par}(\mathsf{p}_1, \top) \\ Y_2 &= \mathsf{par}(\mathsf{p}_2, \mathsf{p}_4, \mathsf{p}_6, \top) & Z_2 &= \mathsf{par}(\mathsf{p}_2, \top) \\ Y_3 &= \mathsf{par}(\mathsf{p}_3, \mathsf{p}_4, \mathsf{p}_5, \mathsf{p}_6, \top) & Z_3 &= \mathsf{par}(\mathsf{p}_3) \end{aligned}$$

Eliminating p_1 from all parity constraints except $\mathsf{par}(\mathsf{p}_1, \top)$:

$$\begin{aligned} Y_1 \oplus Z_1 &= \mathsf{par}(\mathsf{p}_4, \mathsf{p}_5, \top) & Z_1 &= \mathsf{par}(\mathsf{p}_1, \top) \\ Y_2 &= \mathsf{par}(\mathsf{p}_2, \mathsf{p}_4, \mathsf{p}_6, \top) & Z_2 &= \mathsf{par}(\mathsf{p}_2, \top) \\ Y_3 &= \mathsf{par}(\mathsf{p}_3, \mathsf{p}_4, \mathsf{p}_5, \mathsf{p}_6, \top) & Z_3 &= \mathsf{par}(\mathsf{p}_3) \end{aligned}$$

Eliminating p_2 from all parity constraints except $\mathsf{par}(\mathsf{p}_2, \top)$:

$$\begin{aligned} Y_1 \oplus Z_1 &= \mathsf{par}(\mathsf{p}_4, \mathsf{p}_5, \top) & Z_1 &= \mathsf{par}(\mathsf{p}_1, \top) \\ Y_2 \oplus Z_2 &= \mathsf{par}(\mathsf{p}_4, \mathsf{p}_6) & Z_2 &= \mathsf{par}(\mathsf{p}_2, \top) \\ Y_3 &= \mathsf{par}(\mathsf{p}_3, \mathsf{p}_4, \mathsf{p}_5, \mathsf{p}_6, \top) & Z_3 &= \mathsf{par}(\mathsf{p}_3) \end{aligned}$$

Eliminating p_3 from all parity constraints except $\mathsf{par}(\mathsf{p}_3)$:

$$\begin{aligned} Y_1 \oplus Z_1 &= \mathsf{par}(\mathsf{p}_4, \mathsf{p}_5, \top) & Z_1 &= \mathsf{par}(\mathsf{p}_1, \top) \\ Y_2 \oplus Z_2 &= \mathsf{par}(\mathsf{p}_4, \mathsf{p}_6) & Z_2 &= \mathsf{par}(\mathsf{p}_2, \top) \\ Y_3 \oplus Z_3 &= \mathsf{par}(\mathsf{p}_4, \mathsf{p}_5, \mathsf{p}_6, \top) & Z_3 &= \mathsf{par}(\mathsf{p}_3) \end{aligned}$$

Eliminating p_4 from all parity constraints except $\mathsf{par}(\mathsf{p}_4, \mathsf{p}_5, \top)$:

$$\begin{aligned} Y_1 \oplus Z_1 &= \mathsf{par}(\mathsf{p}_4, \mathsf{p}_5, \top) & Z_1 &= \mathsf{par}(\mathsf{p}_1, \top) \\ Y_1 \oplus Y_2 \oplus Z_1 \oplus Z_2 &= \mathsf{par}(\mathsf{p}_5, \mathsf{p}_6, \top) & Z_2 &= \mathsf{par}(\mathsf{p}_2, \top) \\ Y_1 \oplus Y_3 \oplus Z_1 \oplus Z_3 &= \mathsf{par}(\mathsf{p}_6) & Z_3 &= \mathsf{par}(\mathsf{p}_3) \end{aligned}$$

Eliminating p_6 from all parity constraints except $\mathsf{par}(\mathsf{p}_6)$:

$$\begin{aligned} Y_1 \oplus Z_1 &= \mathsf{par}(\mathsf{p}_4, \mathsf{p}_5, \top) & Z_1 &= \mathsf{par}(\mathsf{p}_1, \top) \\ Y_2 \oplus Y_3 \oplus Z_2 \oplus Z_3 &= \mathsf{par}(\mathsf{p}_5, \top) & Z_2 &= \mathsf{par}(\mathsf{p}_2, \top) \\ Y_1 \oplus Y_3 \oplus Z_1 \oplus Z_3 &= \mathsf{par}(\mathsf{p}_6) & Z_3 &= \mathsf{par}(\mathsf{p}_3) \end{aligned}$$

Eliminating p_5 from all parity constraints except $\mathsf{par}(\mathsf{p}_5, \top)$:

$$\begin{aligned} Y_1 \oplus Y_2 \oplus Y_3 \oplus Z_1 \oplus Z_2 \oplus Z_3 &= \mathsf{par}(\mathsf{p}_4) & Z_1 &= \mathsf{par}(\mathsf{p}_1, \top) \\ Y_2 \oplus Y_3 \oplus Z_2 \oplus Z_3 &= \mathsf{par}(\mathsf{p}_5, \top) & Z_2 &= \mathsf{par}(\mathsf{p}_2, \top) \\ Y_1 \oplus Y_3 \oplus Z_1 \oplus Z_3 &= \mathsf{par}(\mathsf{p}_6) & Z_3 &= \mathsf{par}(\mathsf{p}_3) \end{aligned}$$

The obtained affine formula is reduced and semantically equivalent to the input.

The three parity constraints in the left column allow $\neg p_4$, p_5 and $\neg p_6$ to be propagated with the $\xrightarrow[\text{X-unit}]{}$ rule and reach a conflict:

Rule	Assignment ν	Reduct $F\|_\nu$
$\xrightarrow[\text{X-unit}]{}$	$[\neg p_7^\bullet, \neg p_3, p_8, p_1, p_2, p_{13}, p_{10}, \neg p_6]$	$\{\mathsf{or}(\neg p_{12}), \mathsf{or}(\neg p_{11}), \mathsf{or}(p_{11}, p_{12})\} \cup \mathcal{D}(B)\|_\nu$
$\xrightarrow[\text{X-unit}]{}$	$[\neg p_7^\bullet, \neg p_3, p_8, p_1, p_2, p_{13}, p_{10}, \neg p_6, \neg p_4]$	$\{\mathsf{or}(\neg p_{12}), \mathsf{or}(\neg p_{11}), \mathsf{or}(p_{11}, p_{12})\} \cup \mathcal{D}(B)\|_\nu$
$\xrightarrow[\text{X-unit}]{}$	$[\neg p_7^\bullet, \neg p_3, p_8, p_1, p_2, p_{13}, p_{10}, \neg p_6, \neg p_4, p_5]$	$\{\mathsf{or}(\neg p_{12}), \mathsf{or}(\neg p_{11}), \mathsf{or}(p_{11}, p_{12})\} \cup \mathcal{D}(B)\|_\nu$
$\xrightarrow[\text{unit}]{}$	$[\neg p_7^\bullet, \neg p_3, p_8, p_1, p_2, p_{13}, p_{10}, \neg p_6, \neg p_4, p_5, \neg p_{12}]$	$\{\mathsf{or}(\neg p_{11}), \mathsf{or}(p_{11})\} \cup \mathcal{D}(B)\|_\nu$
$\xrightarrow[\text{unit}]{}$	$[\neg p_7^\bullet, \neg p_3, p_8, p_1, p_2, p_{13}, p_{10}, \neg p_6, \neg p_4, p_5, \neg p_{12}, \neg p_{11}]$	$\{\mathsf{par}(\emptyset)\} \cup \mathcal{D}(B)\|_\nu$

■

The second technique is related to clause learning. In the classical conflict analysis framework explained in Section 4.1.3 conflicts are reached when the empty clause $\mathsf{or}(\emptyset)$ is in the current reduct. A classically compatible implication graph can then be constructed using clauses that provoked unit propagation as reason clauses. Furthermore, the clause in F that becomes $\mathsf{or}(\emptyset)$ in the reduct $F|_\nu$ acts as a reason clause for $\bot$. However, the use of Gauss-Jordan elimination allows some useful extensions in this framework. Firstly, the $\xrightarrow[\text{X-unit}]{}$ rule now allows to propagate literals that lack of a classical reason clause. Secondly, conflicts now may be detected by finding that $A \cup \mathrm{xor}(\nu)$ is unsatisfiable for some parity formula with $\mathcal{D}(A) \subseteq F$, rather than finding the empty clause $\mathsf{or}(\emptyset)$ in the reduct of F. In both cases, Gauss-Jordan elimination is used to either find entailed literals or detect unsatisfiability, and therefore the problem boils down to find if $A \cup \mathrm{xor}(\nu) \models \mathrm{xor}(\lambda)$, where λ is a pseudoliteral.[1]

Observe now that Gauss-Jordan elimination uses exclusively parity constraint addition to derive a reduced affine formula, which means that $A \cup \mathrm{xor}(\nu) \models \mathrm{xor}(\lambda)$ if and only if there are literals $l_1, \ldots, l_n$ occurring in ν and a subset $A' \subseteq A$ such that:

$$\bigoplus_{X \in A'} X \oplus \bigoplus_{i=1}^{n} \mathrm{xor}(l_i) = \mathrm{xor}(\lambda)$$

By Proposition 2.22, this is equivalent to:

$$\bigoplus_{X \in A'} X = \mathrm{xor}(\lambda) \oplus \bigoplus_{i=1}^{n} \mathrm{xor}(l_i) \tag{5.1}$$

Observe that the parity constraint (5.1) can be expressed as an addition of some parity constraints in A, which means that it is a semantic consequence of F. Such parity constraints are called *parity explanations* in [LJN12, LJN14b, Lai14], and are the basis of the clause learning method explained there. We present a version of these methods adapted to our framework, in which such parity explanations are called *parity constraint reasons*.

Definition 5.15 (XOR-compatible implication graphs):
Let K be an implication graph and λ be a non-assumption vertex in K. The reason parity constraint *for λ on K is*

[1] It might be worth at this point remembering that $\mathrm{xor}(\bot) = \mathsf{par}(\top)$, see (4.2) on page 44; and $\mathrm{xor}(\nu) = \{\mathrm{xor}(l) : l \in \mathrm{lits}(\nu)\}$, see Definition 4.1 on page 4.1.

the parity constraint:

$$\mathcal{X}(\lambda) \quad = \bigoplus_{\mu \text{ precedes } \lambda \text{ in } K} \mathrm{xor}(\mu) \oplus \mathrm{xor}(\lambda)$$

The implication graph K is said to be XOR-compatible *with a state $\langle F, \nu \rangle$ if the following hold:*

i) K contains as vertices exactly the literals occurring in ν without decision decoration, together with the $\bot$ symbol.

ii) Assumption vertices in K are exactly those literals occurring as decision literals in K.

iii) For every non-assumption literal λ, either of the following holds:

a) The reason clause $\mathcal{R}(\lambda)$ is included in F.

b) There is an affine formula A such that $\mathcal{D}(A) \subseteq F$ and the reason parity constraint $\mathcal{X}(\lambda)$ is $\bigoplus_{X \in A} X$.

We now show that our framework is sound, i.e. clauses learned from XOR-compatible implication graphs are semantic consequences of the CNF formula in the solver state.

Lemma 5.16 (Reason parity constraints imply reason clauses):
Let K be an implication graph and λ be a non-assumption vertex in K. Then, the reason clause $\mathcal{R}(\lambda)$ for λ occurs in the direct encoding $\mathcal{D}(\mathcal{X}(\lambda))$ of the reason parity constraint for λ.

Proof. Let $U = \{\mathrm{var}(\mu) : \mu \text{ precedes } \lambda\}$, which is well-defined because $\bot$ precedes no pseudoliteral in K. Consider the set of variables:

$$V = \{\mathrm{var}(\mu) : \mu \text{ is a positive literal and } \mu \text{ precedes } \lambda \text{ in } K\} \tag{5.2}$$

We first show that the clause $C = \mathsf{or}(\lambda) \vee \mathrm{nand}(V) \vee \mathsf{or}(U \setminus V)$ is the reason clause $\mathcal{R}(\lambda)$, which is defined as:

$$\mathcal{R}(\lambda) = \mathsf{or}(\lambda) \vee \mathsf{or}(\overline{\mu} : \mu \text{ precedes } \lambda \text{ in } K) \tag{5.3}$$

Showing that $\mathcal{R}(\lambda) = C$ is then equivalent to showing that positive literals in $\mathsf{or}(\overline{\mu} : \mu \text{ precedes } \lambda \text{ in } K)$ have as underlying variables exactly those in $U \setminus V$; and similarly, negative literals have as underlying variables exactly those in V. Both follow from (5.2) and (5.3) above by observing that, if μ is a positive literal, then $\overline{\mu} = \overline{\mathrm{var}(\mu)}$; and if μ is a negative literal, then $\overline{\mu} = \mathrm{var}(\mu)$. Hence, $\mathcal{R}(\lambda) = C$ holds.

We now show that the number of negative literals in C is congruent modulo 2 to $\mathrm{pol}(\mathcal{X}(\lambda)) + 1$. Let us denote $L = \{\mu : \mu \text{ precedes } \lambda\}$. We distinguish three cases:

- If μ is a positive literal, then the polarity of $\mathcal{X}(\lambda) = \bigoplus_{\mu \in L} \mathrm{xor}(\mu) \oplus \mathsf{par}(\lambda, \top)$ verifies:

  $$\mathrm{pol}(\mathcal{X}(\lambda)) \approx \sum_{\mu \in L} \mathrm{pol}(\mathrm{xor}(\mu)) + 1 = |\{\mu \in L : \mu \text{ is positive}\}| + 1 = |V| + 1$$

  Observe that in this case negative literals in C have underlying variables corresponding precisely to variables in V. Setting $V' = V$ yields $|V'| \approx \mathrm{pol}(\mathcal{X}(\lambda)) + 1$.

- If μ is a negative literal, then the polarity of $\mathcal{X}(\lambda) = \bigoplus_{\mu \in L} \mathrm{xor}(\mu) \oplus \mathsf{par}(\lambda)$ verifies:

  $$\mathrm{pol}(\mathcal{X}(\lambda)) \approx \sum_{\mu \in L} \mathrm{pol}(\mathrm{xor}(\mu)) = |\{\mu \in L : \mu \text{ is positive}\}| = |V|$$

  In this case, negative literals in C have variables in $V \cup \{\mathrm{var}(\lambda)\}$ as underlying variables. In particular, setting $V' = V \cup \{\mathrm{var}(\lambda)\}$ yields $|V'| \approx \mathrm{pol}(\mathcal{X}(\lambda)) + 1$.

- If $\mu = \bot$, then the polarity of $\mathcal{X}(\lambda) = \mathcal{X}(\lambda) = \bigoplus_{\mu \in L} \mathrm{xor}(\mu) \oplus \mathsf{par}(\bot) = \bigoplus_{\mu \in L} \mathrm{xor}(\mu) \oplus \mathsf{par}(\top)$ verifies:

  $$\mathrm{pol}(\mathcal{X}(\lambda)) \approx \sum_{\mu \in L} \mathrm{pol}(\mathrm{xor}(\mu)) + 1 = |\{\mu \in L : \mu \text{ is positive}\}| + 1 = |V| + 1$$

In this case, negative literals in C have variables in V as underlying variables. In particular, setting $V' = V$ yields $|V'| \approx \text{pol}(\mathcal{X}(\lambda)) + 1$.

Now consider the set of variables $U' = \text{vars}(\mathcal{X}(\lambda))$. Then, it is straightforward to check that

$$C = \text{or}(\lambda) \vee \text{nand}(V) \vee \text{or}(U \setminus V) = \text{nand}(V') \vee \text{or}(U' \setminus V')$$

and therefore we conclude that $C = \mathcal{R}(\lambda) \in \mathcal{D}(\mathcal{X}(\lambda))$. □

Corollary 5.17 (Entailment-soundness of parity reasoning-based clause learning):
Let F be a CNF formula, ν be a partial assignment and K be an implication graph XOR-compatible with $\langle F, \nu \rangle$. Let σ be a shift on K. Then, $F \models \mathcal{C}(\sigma)$.

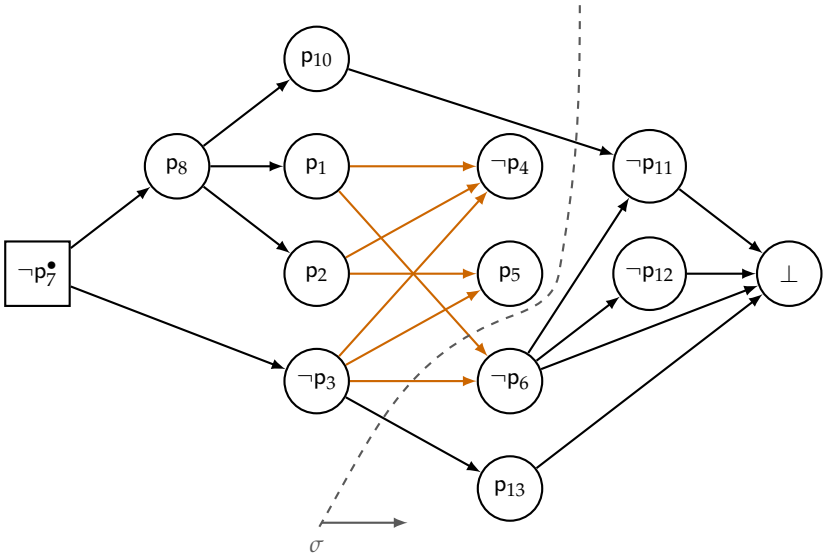
Proof. Let $G = \{\mathcal{R}(\lambda) : \lambda \text{ is a non-assumption vertex in } K\}$. We show that $F \models G$. Let λ be a non-assumption vertex in K. By Definition 5.15, either of the following hold:

- $\mathcal{R}(\lambda) \in F$. In this case, $F \models \mathcal{R}(\lambda)$ immediately.
- There is an affine formula A such that $\mathcal{D}(A) \subseteq F$ and $\mathcal{X}(\lambda) = \bigoplus_{X \in A} X$. In this case, observe that, since parity constraint addition is sound, we obtain $A \models \mathcal{X}(\lambda)$. Now, by Proposition 5.2, $A \equiv \mathcal{D}(A)$ and $\mathcal{X}(\lambda) \equiv \mathcal{D}(\mathcal{X}(\lambda))$, and so we conclude $F \models \mathcal{D}(\mathcal{X}(\lambda))$. In particular, Lemma 5.16 implies that $F \models \mathcal{R}(\lambda)$.

Since λ was chosen arbitrarily, $F \models G$. Now, G satisfies the conditions in Theorem 4.8 so that $\mathcal{C}(\sigma)$ is a linear resolvent in G. In particular, $G \models \mathcal{C}(\sigma)$, and the claim follows. □

Corollary 5.17 shows that it is sound to introduce learned clauses from implication graphs whenever the latter are XOR-compatible with the current state in the SAT solver. This justifies the $\xrightarrow[\text{X-learn}]{}$ rule in Table 5.2. Let us ilustrate with an example how conflict analysis works in practice.

Example 5.18 (Interleaved parity reasoning: clause learning):
We continue here with Example 5.14. Let $\langle F, \nu \rangle$ be the last state of the SAT solver there. We can try to build a implication graph for the conflict found there, shown as the black-colored part of the following implication graph:



Observe that literals $\neg p_4$, p_5 and $\neg p_6$ do not have reasons clauses in F, and so this implication graph is not clasically compatible with $\langle F, \nu \rangle$. We can nevertheless obtain reason clauses for them that do not belong to F, but are semantic consequences of it, represented by brown arrows. Let us show how can this be done.

As shown in the Gauss-Jordan elimination execution in Example 5.14, we can express those literals as follows:

$$\begin{aligned}
\neg p_4 = \mathsf{par}(p_4) \quad &= Y_1 \oplus Y_2 \oplus Y_3 \oplus Z_1 \oplus Z_2 \oplus Z_3 = (Y_1 \oplus Y_2 \oplus Y_3) \oplus (\mathrm{xor}(p_1) \oplus \mathrm{xor}(p_2) \oplus \mathrm{xor}(\neg p_3)) \\
p_5 = \mathsf{par}(p_5, \top) &= Y_2 \oplus Y_3 \oplus Z_2 \oplus Z_3 \qquad = (Y_2 \oplus Y_3) \oplus (\mathrm{xor}(p_2) \oplus \mathrm{xor}(\neg p_3)) \\
\neg p_6 = \mathsf{par}(p_6) \quad &= Y_1 \oplus Y_3 \oplus Z_1 \oplus Z_3 \qquad = (Y_1 \oplus Y_3) \oplus (\mathrm{xor}(p_1) \oplus \mathrm{xor}(\neg p_3))
\end{aligned}$$

We can reorder the additions to obtain:

$$\begin{aligned}
Y_1 \oplus Y_2 \oplus Y_3 &= \mathrm{xor}(\neg p_4) \oplus \mathrm{xor}(p_1) \oplus \mathrm{xor}(p_2) \oplus \mathrm{xor}(\neg p_3) = \mathsf{par}(p_1, p_2, p_3, p_4) = D \\
Y_2 \oplus Y_3 &= \mathrm{xor}(p_5) \oplus \mathrm{xor}(p_2) \oplus \mathrm{xor}(\neg p_3) \qquad = \mathsf{par}(p_2, p_3, p_5) = D' \\
Y_1 \oplus Y_3 &= \mathrm{xor}(\neg p_6) \oplus \mathrm{xor}(p_1) \oplus \mathrm{xor}(\neg p_3) \qquad = \mathsf{par}(p_1, p_3, p_6, \top) = D''
\end{aligned} \tag{5.4}$$

The left hand sides are additions in A, so in particular the right hand sides above are reason parity constraints for literals $\neg p_4$, p_5 and $\neg p_6$, and semantic consequences of F. Then, we have the following choice of reason clauses and reason parity constraints:

Vertex	Reason clause/parity constraint
p_1	$\mathcal{R}(p_1) = C_{29} = \mathsf{or}(p_1, \neg p_8)$
p_2	$\mathcal{R}(p_2) = C_{30} = \mathsf{or}(p_2, \neg p_8)$
$\neg p_3$	$\mathcal{R}(\neg p_3) = C_{32} = \mathsf{or}(\neg p_3, p_7)$
$\neg p_4$	$\mathcal{X}(\neg p_4) = D = \mathsf{par}(p_1, p_2, p_3, p_4)$
p_5	$\mathcal{X}(p_5) = D' = \mathsf{par}(p_2, p_3, p_5)$
$\neg p_6$	$\mathcal{X}(\neg p_6) = D'' = \mathsf{par}(p_1, p_3, p_6, \top)$
$\neg p_7$	none (decision literal)
p_8	$\mathcal{R}(p_8) = C_{37} = \mathsf{or}(p_7, p_8)$
p_{10}	$\mathcal{R}(p_{10}) = C_{39} = \mathsf{or}(\neg p_8, p_{10})$
$\neg p_{11}$	$\mathcal{R}(\neg p_{11}) = C_{34} = \mathsf{or}(p_6, \neg p_{10}, \neg p_{11})$
$\neg p_{12}$	$\mathcal{R}(\neg p_{12}) = C_{33} = \mathsf{or}(p_6, \neg p_{12})$
p_{13}	$\mathcal{R}(p_{13}) = C_{31} = \mathsf{or}(p_3, p_{13})$
$\bot$	$\mathcal{R}(\bot) = C_{35} = \mathsf{or}(p_6, p_{11}, p_{12}, \neg p_{13})$

This makes the implication graph at the begining of this example XOR-compatible with the last state in the solver. Observe that reason clauses for $\xrightarrow[\text{X-unit}]{}$-propagated literals $\neg p_4$, p_5 and $\neg p_6$ in the implication graph occur in the direct encoding of their reason parity constraints:

$$\begin{aligned}
\mathcal{R}(\neg p_4) = \mathsf{or}(\neg p_1, \neg p_2, p_3, \neg p_4) \in \mathcal{D}(\mathsf{par}(p_1, p_2, p_3, p_4)) = \mathcal{D}(\mathcal{X}(\neg p_4)) \\
\mathcal{R}(p_5) = \mathsf{or}(\neg p_2, p_3, p_5) \in \mathcal{D}(\mathsf{par}(p_2, p_3, p_5)) = \mathcal{D}(\mathcal{X}(p_5)) \\
\mathcal{R}(\neg p_6) = \mathsf{or}(\neg p_1, p_3, \neg p_6) \in \mathcal{D}(\mathsf{par}(p_1, p_3, p_6, \top)) = \mathcal{D}(\mathcal{X}(\neg p_6))
\end{aligned}$$

This means that the clauses in the left hand side are semantic consequences of F, because so are the parity constraints in the right. The shift $\sigma = [\bot, \neg p_{11}, \neg p_{12}, p_{13}, \neg p_6]$ shown in the implication graph generates a learned clause $\mathcal{C}(\sigma) = \mathsf{or}(\neg p_1, p_3, \neg p_{10})$. As in Example 4.13 we can obtain the learned clause as a linear resolvent in $F \cup \{\mathcal{X}(\neg p_4), \mathcal{X}(p_5), \mathcal{X}(\neg p_6)\}$, therefore $F \models \mathcal{C}(\sigma)$:

$$\begin{aligned}
\mathsf{or}(\mathsf{p}_6, \mathsf{p}_{11}, \mathsf{p}_{12}, \neg\mathsf{p}_{13}) &= \mathcal{R}(\bot) \\
\mathsf{or}(\mathsf{p}_6, \neg\mathsf{p}_{10}, \mathsf{p}_{12}, \neg\mathsf{p}_{13}) &= \mathcal{R}(\bot) \otimes_{\mathsf{p}_{11}} \mathcal{R}(\neg\mathsf{p}_{11}) \\
\mathsf{or}(\mathsf{p}_6, \neg\mathsf{p}_{10}, \neg\mathsf{p}_{13}) &= \mathcal{R}(\bot) \otimes_{\mathsf{p}_{11}} \mathcal{R}(\neg\mathsf{p}_{11}) \otimes_{\mathsf{p}_{12}} \mathcal{R}(\neg\mathsf{p}_{12}) \\
\mathsf{or}(\mathsf{p}_3, \mathsf{p}_6, \neg\mathsf{p}_{10}) &= \mathcal{R}(\bot) \otimes_{\mathsf{p}_{11}} \mathcal{R}(\neg\mathsf{p}_{11}) \otimes_{\mathsf{p}_{12}} \mathcal{R}(\neg\mathsf{p}_{12}) \otimes_{\neg\mathsf{p}_{13}} \mathcal{R}(\mathsf{p}_{13}) \\
\mathsf{or}(\neg\mathsf{p}_1, \mathsf{p}_3, \neg\mathsf{p}_{10}) &= \mathcal{R}(\bot) \otimes_{\mathsf{p}_{11}} \mathcal{R}(\neg\mathsf{p}_{11}) \otimes_{\mathsf{p}_{12}} \mathcal{R}(\neg\mathsf{p}_{12}) \otimes_{\neg\mathsf{p}_{13}} \mathcal{R}(\mathsf{p}_{13}) \otimes_{\mathsf{p}_6} \mathcal{R}(\neg\mathsf{p}_6) = \mathcal{C}(\sigma)
\end{aligned}$$

In particular, we can introduce the clause $\mathsf{or}(\neg\mathsf{p}_1, \mathsf{p}_3, \neg\mathsf{p}_{10})$ in F by the transition $\langle F, \nu\rangle \xrightarrow[\text{X-learn}]{} \langle F \cup \{\mathcal{C}(\sigma)\}, \nu\rangle$ in the SAT solver. ■

We model the execution of CDCL-style SAT solving with integrated parity reasoning, which we refer to as *XOR-CDCL SAT solving*, by the transition rule $\xrightarrow[\text{X-CDCL}]{}$, defined as the union of the following transition relations:

$$\xrightarrow[\text{unit}]{} \qquad \xrightarrow[\text{decide}]{} \qquad \xrightarrow[\text{back}]{} \qquad \xrightarrow[\text{X-unit}]{} \qquad \xrightarrow[\text{X-learn}]{} \qquad \xrightarrow[\text{X-simp}]{}$$

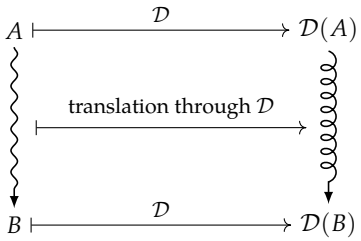
5.4 Contributions

In this chapter, applications of parity reasoning to SAT solving have been discussed, following the inprocessing techniques proposed by Soos, Nohl and Castelluccia [SNC09, Soo12] and by Laitinen, Junttila and Niemelä [Lai14, LJN12, LJN14b]. Their methods have been formalized under frameworks defined in previous chapters:

- Gauss-Jordan elimination over affine formulae has been shown to be a proving procedure with the **Xor** proof system presented in Section 3.3 as underlying proof system. As a consequence, we are able to extract **Xor**-derivations from the execution of the Gauss-Jordan elimination algorithm as shown in Algorithm 1 and Theorem 5.12.
- We have defined an extension to the classical CDCL transition system from Section 4.1, called XOR-CDCL, which models state-of-the-art techniques for SAT solvers with integrated parity reasoning through Gauss-Jordan elimination. The rule $\xrightarrow[\text{X-simp}]{}$ presented in Section 5.3.1 covers inprocessing parity reasoning, where affine formulae encoded in a CNF formula are simplified and then encoded back into CNF. The rule $\xrightarrow[\text{X-unit}]{}$ allows propagation of entailed literals that can be detected by Gauss-Jordan elimination. With a suitable modification to keep track of added parity constraints during parity reasoning, implication graphs can be constructed upon a conflict. Clause learning from such graphs covered by the $\xrightarrow[\text{X-learn}]{}$ rule. The latter two rules provide a formalization for interleaved parity reasoning.
- Implication graphs generated during interleaved parity reasoning have been introduced as instances of the generalized conflict analysis framework defined in Section 4.1.2, called XOR-compatible implication graphs. Clause learning over them is sound by Theorem 5.17.

Modeling parity reasoning in SAT solving with instances of the general frameworks defined in previous chapters allows the immediate adaption of the unsatisfiability proof generation schema in Section 4.2.2. In particular, to generate unsatisfiability proofs for XOR-CDCL SAT solving we need to provide, on one hand, **Rat**-derivations for applications of the $\xrightarrow[\text{X-simp}]{}$ rule; and on the other hand, **Rat**-derivations of parity constraint reasons in XOR-compatible implication graphs. In practice, the methods above are applied using Gauss-Jordan elimination or a restriction thereof, thus the availability of a **Xor**-derivation of the affine correspondents can be assumed. Hence, as further developed in Section 8.2, the problem of generating

unsatisfiability proofs for the XOR-CDCL system is then reduced to *translate* **Xor**-*derivations into* **Rat**-*derivations through the direct encoding*. That is, given a **Xor**-derivation of an affine formula B from another affine formula A, we need to obtain a **Rat**-derivation of the direct encoding of B from the direct encoding of A:



Part II is devoted to obtain translations through the direct encoding of **Xor**-derivations. We present two translations. Direct translations translate **Xor**-derivations inference-wise, whereas linear translations use an intermediate translation using splittings, which constitute a minor contribution in this chapter as a regular version of the natural splitting developed by Gwynne and Kullman [GK13].

Part II

Proof translations through the direct encoding

Chapter 6

Direct translation of extended XOR derivations

Part I was concluded by justifying the need for translations of **Xor**-derivations into **Rat**-derivations through the direct encoding. Attaining this provides, together with our previous results, a method for generation of unsatisfiability proofs for the XOR-CDCL SAT solving procedure described in Section 5.3. In this chapter, we present a method to generate such translations, which we call *direct translation*. Direct translations are constructed by building independent **Rat**-derivations that translate every inference along an **Xor**-derivation. This is, at least theoretically, enough to solve our research question.

Nonetheless, direct translations are critically flawed: translated **Xor** inferences are exponentially-sized in the size of parity constraints involved in the inference. Despite this weakness, there are compelling reasons to introduce direct translations. Above all, direct translations are a fundamental tool in the development of a second translation, called *linear translation*, which is the subject of Chapter 7. Linear translations are far more tractable than direct translations, at least from an asymptotical, worse-case perspective. A second reason is that our empirical evaluation of direct and linear translations shows that direct translations overperform linear translations in some practical situations. A compared study of translation lengths is carried out in Section 8.1.

To define linear translations, translations of **EXor**-derivations (which may in addition include XOR definition introduction inferences) through the direct encoding are required. For that reason, in this chapter we consider **EXor**-derivations as input. Since **Xor**-derivations are also **EXor**-derivations, this does not affect the use of direct translations for unsatisfiability proof generation.

This chapter is divided in three sections. Since parity constraint addition inferences are significantly more complex to translate than the other inferences in the **EXor** proof system, they are considered separately in Section 6.1. The remaining inferences, namely parity constraint deletion and XOR definition introduction, are provided translations in Section 6.2, together with the construction of the full translation. Before direct translations are formally defined, we outline the construction process. Finally, our results are discussed in Section 6.3.

An outline of direct translations. Direct translations are our main interface between the **EXor** proof system and the **Rat** proof system. The direct translation maps every **EXor**-derivation φ into a translation $\Delta(\varphi)$ through the direct encoding. This means that $\Delta(\varphi)$ is a **Rat**-derivation with the direct encoding of premises of φ as premises, and the direct encoding of conclusions of φ as conclusions.

Our construction is of inductive nature. Every inference along the original **EXor**-derivation is translated into a **Rat**-derivation, as shown in Figure 6.1. Along the following sections, diagrams like this will be used to conceptually depict how translations will be greatly helpful. A "wave" arrow from A to B denotes an **EXor**-derivation of B from A; a "coil" arrow from F to G denotes a **Rat**-derivation of G from F. A "$\Rightarrow$" arrow, as usual throughout this Thesis, denotes an inference. "$\mapsto$" arrows denote the application of a mapping.

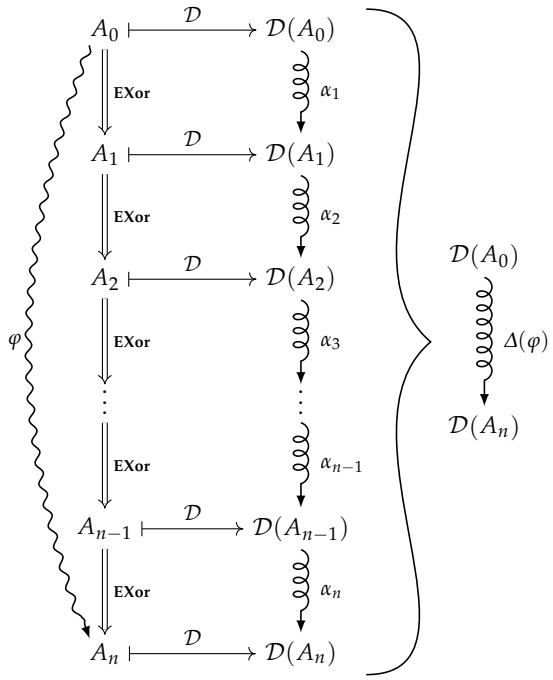


Figure 6.1: Direct translation of **EXor**-derivations. The **EXor**-derivation φ of the affine formula B from the affine formula A is translated into a **Rat**-derivation $\Delta(\varphi)$ of the direct encoding of B from the direct encoding of A. Every inference $A_{i-1} \Rightarrow_{\mathbf{EXor}} A_i$ along the derivation φ is translated independently into a **Rat**-derivation α_i of the direct encoding of A_i from the direct encoding of A_{i-1}. Concatenation, represented by a brace, of the α_i yields the direct translation $\Delta(\varphi)$.

In this case, the mapping $\mathcal{D}$ maps every affine formula A to its direct encoding $\mathcal{D}(A)$. This is an example of a mapping that acts over formulae. Since we will be working with translations, we will consider mappings acting over derivations as well (e.g. see Figure 6.4 in page 83). Finally, we consider two kinds of brackets. Braces "}" denote concatenation of derivations, where as brackets "]" denote fusion of derivations (e.g in Figure 7.2 in page 98).

In Figure 6.1, single inferences in φ are translated through the direct encoding into **Rat**-derivations. This means that every inference $A_{i-1} \Rightarrow_{\mathbf{EXor}} A_i$ is mapped to a **Rat**-derivation of $\mathcal{D}(A_i)$ from $\mathcal{D}(A_{i-1})$. Since the obtained derivations have matching ends, they can be concatenated, resulting in the direct translation $\Delta(\varphi)$ of φ. To translate **EXor** inferences, every rule in Definition 3.23 must be provided a translation. Translations for parity constraint addition inferences are presented in Definition 6.8. XOR definition introduction inferences are translated in Definition 6.11, and finally parity constraint deletion inferences are assigned a translation in Definition 6.13.

6.1 Direct translation of addition inferences

We now tackle the problem of generating a partial translation for parity constraint addition inferences. That is, given an affine formula A and two parity constraints $Y, Z \in A$, the parity constraint addition inference $A \Rightarrow_{\mathbf{EXor}} A \cup \{Y \oplus Z\}$ holds. We aim to obtain a translation of this inference through the direct encoding. We obtain a **At**-derivation $\Delta_{\text{add}}(A, Y, Z)$ of the direct encoding of $A \cup \{Y \oplus Z\}$ from the direct encoding of A. However, our general construction does not work for a few degenerate cases. Ad hoc derivations are given below for those.

Proposition 6.1 (Correctness of direct translation of addition inferences, degenerate case):
Let A be an affine formula and $Y, Z \in A$ be two parity constraints. Define a string α of CNF formulae for the following cases:

i) If $\mathrm{vars}(Y)$ *and* $\mathrm{vars}(Z)$ *are disjoint, then let α be the string* $[\mathcal{D}(A)\,,\, \mathcal{D}(A \cup \{Y \oplus Z\})]$.

ii) If $Y \oplus Z \in A$, then let α be the string $[\mathcal{D}(A)]$.

iii) If $Y = Z$, then let α be the string $[\mathcal{D}(A)]$.

Then, the string α is an **At***-derivation of $\mathcal{D}(A)$ from $\mathcal{D}(A \cup \{Y \oplus Z\})$.*
Proof.

i) Let X be the addition $Y \oplus Z$, and consider an arbitrary clause $C \in \mathcal{D}(X)$. Then, C is of the form $\mathrm{nand}(V) \vee \mathsf{or}(\mathrm{vars}(X) \setminus V)$ for some set of variables

$$V \subseteq \mathrm{vars}(X) = \mathrm{vars}(Y) \triangle \mathrm{vars}(Z) = \mathrm{vars}(Y) \cup \mathrm{vars}(Z)$$

such that $|V| \approx \mathrm{pol}(X) + 1$. Define the sets of variables $V_Y = V \cap \mathrm{vars}(Y)$ and $V_Z = V \cap \mathrm{vars}(Z)$. Since Y and Z are disjoint, we have $|V| = |V_Y| + |V_Z|$. On the other hand, the polarity of X verifies $\mathrm{pol}(X) \approx \mathrm{pol}(Y) + \mathrm{pol}(Z)$. Therefore, we obtain $|V_Y| + |V_Z| \approx \mathrm{pol}(Y) + \mathrm{pol}(Z) + 1$, or equivalently, $|V_Y| + |V_Z| \not\approx \mathrm{pol}(Y) + \mathrm{pol}(Z)$. By Proposition 2.1 (page 9), we conclude that exactly one of the following holds:

$$|V_Y| \not\approx \mathrm{pol}(Y) \qquad\qquad |V_Z| \not\approx \mathrm{pol}(Z)$$

Let us assume that $|V_Y| \not\approx \mathrm{pol}(Y)$ holds. Consider the clause $D = \mathrm{nand}(V_Y) \vee \mathsf{or}(\mathrm{vars}(Y) \setminus V_Y)$. From $V_Y \subseteq V$ and $\mathrm{vars}(Y) \setminus V_Y \subseteq \mathrm{vars}(X) \setminus V$, we conclude that all literals in D occur in C as well, so D subsumes $C \in \mathcal{D}(Y)$. By Proposition 3.17, C is an asymmetric tautology in $\mathcal{D}(Y)$, therefore an asymmetric tautology in $\mathcal{D}(A)$.

A similar argument for the case when $|V_Z| \not\approx \mathrm{pol}(Z)$ holds concludes that C is an asymmetric tautology in $\mathcal{D}(Z)$, hence in $\mathcal{D}(A)$. All cases for C are covered by these arguments, which shows that $\mathcal{D}(X) \subseteq \mathcal{D}(A)^{\mathsf{AT}}$. Hence, $\mathcal{D}(A) \Rightarrow_{\mathbf{At}} \mathcal{D}(A \cup \{X\})$ and then the claim follows.

ii) Follows straightforward from observing that, if $Y \oplus Z \in A$, then $\mathcal{D}(A \cup \{Y \oplus Z\}) = \mathcal{D}(A)$.

iii) If $Y = Z$, then their addition is $Y \oplus Z = \mathsf{par}(\emptyset)$, whose direct encoding is empty. Then we obtain $\mathcal{D}(A \cup \{Y \oplus Z\}) = \mathcal{D}(A) \cup \mathcal{D}(\mathsf{par}(\emptyset)) = \mathcal{D}(A)$, and the claim follows. □

The cases above allow the generation of translations for addition inferences when either antecedents are equal, or their variable sets are disjoint, or their addition is redundant. We now tackle all other cases: we assume that $\mathrm{vars}(Y) \cap \mathrm{vars}(Z) \neq \emptyset$ and $\mathsf{par}(\emptyset) \neq Y \oplus Z \notin A$. Let us first give an intuition of how the translation can be obtained.

An outline of the translation. The derivation consists on a series of resolution steps, and therefore it can be regarded as a resolution tree, i.e. a tree with nodes labeled by clauses where every non-root node is the resolvent of its two child nodes. Recall that every linear resolvent can be obtained as an asymmetric tautology (see Proposition 3.16), and in particular so does every resolvent. It is somewhat easier to understand the construction in a bottom-up fashion. That is, we start from a clause $C \in \mathcal{D}(Y \oplus Z)$ in the bottom level of the resolution tree, and we iteratively find premises that yield clauses in the current top level as resolvents. The constructed tree can be regarded as an **At**-derivation of the clause C from the top level clauses. If we find that all top level clauses are included in the direct encoding of $\{Y, Z\}$, then we will have an **At**-derivation of a CNF formula containing C from $\mathcal{D}(A)$.

$$\dfrac{\dfrac{\dfrac{\ddots \quad \iddots}{C \vee \mathsf{or}(a_1, a_2)} \otimes \quad \dfrac{\ddots \quad \iddots}{C \vee \mathsf{or}(a_1, \overline{a_2})} \otimes}{C \vee \mathsf{or}(a_1)} \otimes \quad \dfrac{\dfrac{\ddots \quad \iddots}{C \vee \mathsf{or}(\overline{a_1}, a_2)} \otimes \quad \dfrac{\ddots \quad \iddots}{C \vee \mathsf{or}(\overline{a_1}, \overline{a_2})} \otimes}{C \vee \mathsf{or}(\overline{a_1})} \otimes}{C} \otimes$$

Figure 6.2: Bottom part of the resolution tree for the direct translation of an addition inference. The clause C is in the zeroth level. Clauses $C \vee \mathsf{or}(a_1)$ and $C \vee \mathsf{or}(\overline{a_1})$ are in the first level. Clauses $C \vee \mathsf{or}(a_1, a_2)$, $C \vee \mathsf{or}(\overline{a_1}, a_2)$, $C \vee \mathsf{or}(a_1, \overline{a_2})$ and $C \vee \mathsf{or}(\overline{a_1}, \overline{a_2})$ are in the second level.

Assume that common variables to Y and Z are enumerated as $V = \mathrm{vars}(Y) \cap \mathrm{vars}(Z) = \{a_1, \ldots, a_n\}$. Observe that C contains all variables in $Y \oplus Z$, i.e. variables in $\mathrm{vars}(Y) \triangle \mathrm{vars}(Z)$, as either positive or negative literals, because C is a clause in the direct encoding of $Y \oplus Z$. We start with C in the bottom zeroth level of our resolution tree. Our construction iteratively adds a literal with underlying variable a_j to each clause in the $(j-1)$-th level to obtain the clauses in the j-th level. This is done for both literals a_j and $\neg a_j$, so for every clause D in the $(j-1)$-th level, the j-th level contains both $D \vee \mathsf{or}(a_j)$ and $D \vee \mathsf{or}(\neg a_j)$, for resolving this two clauses upon a_j yields D. Iterating this process results in the resolution tree shown in Figure 6.2.

This process is stopped when the $(n-1)$-th level is generated. At this point, clauses in the $(n-1)$-th level are of the form $C \vee \mathsf{or}(l_1, \ldots, l_{n-1})$, where the literals l_j have underlying variable $\mathrm{var}(l_j) = a_j$. The subclause C in each of those clauses can be splitted as the disjunction of two clauses C_Y and C_Z such that:

$$\mathrm{vars}(C_Y) = \mathrm{vars}(Y) \setminus \mathrm{vars}(Z) \qquad \mathrm{vars}(C_Z) = \mathrm{vars}(Z) \setminus \mathrm{vars}(Y)$$

This is, every underlying variable in C comes from exactly one parity constraint among Y and Z, and consequently C_Y and C_Z separate literals in C according to whether their underlying variables originate in Y or Z. By parity considerations (see the proof of Lemma 6.4), there is a choice from $l_n = a_n$ or $l_n = \neg a_n$ with the property that the clause $C_Y \vee \mathsf{or}(l_1, \ldots, l_n)$ lies in the direct encoding of Y and, correspondingly, $C_Z \vee \mathsf{or}(l_1, \ldots, l_n)$ occurs in the direct encoding of Z. We can then complete the top of the resolution tree in Figure 6.2 by appending the following resolution subtree on top of every clause $C \vee \mathsf{or}(l_1, \ldots, l_{n-1})$ in the top level:

$$\dfrac{C_Y \vee \mathsf{or}(l_1, \ldots, l_{n-1}, l_n) \qquad C_Z \vee \mathsf{or}(l_1, \ldots, l_{n-1}, \overline{l_n})}{C \vee \mathsf{or}(l_1, \ldots, l_{n-1})} \otimes$$

The resulting tree has C as root node and all leaves in either $\mathcal{D}(Y)$ or $\mathcal{D}(Z)$. This can be done for every clause C in the direct encoding of $Y \oplus Z$, and we can regard these trees as a derivation of $\mathcal{D}(A \cup \{Y \oplus Z\})$ from $\mathcal{D}(A)$. In practice, trees for every clause will be generated simultaneously, so first we derive the $(n-1)$-th level of all trees, then the $(n-2)$-th level of all trees, and so forth until the 0-th level of all trees is derived. The latter contains all clauses in the direct encoding of $Y \oplus Z$.

Example 6.2 (Direct translation of addition inferences):
Consider parity constraints X, Y and Z with $X = Y \oplus Z$:

$$X = \mathsf{par}(\mathsf{p}_4, \mathsf{p}_5, \top) \qquad Y = \mathsf{par}(\mathsf{p}_1, \mathsf{p}_2, \mathsf{p}_3, \mathsf{p}_4) \qquad Z = \mathsf{par}(\mathsf{p}_1, \mathsf{p}_2, \mathsf{p}_3, \mathsf{p}_5, \top)$$

The direct encoding of X is given by $\mathcal{D}(X) = \{\mathsf{or}(\mathsf{p}_4, \mathsf{p}_5)\,, \mathsf{or}(\neg\mathsf{p}_4, \neg\mathsf{p}_5)\}$. As shown in Figure 6.3, both clauses in $\mathcal{D}(X)$ can be obtained by resolution from $\mathcal{D}(\{Y, Z\})$. ■

Constructing the translation. We now present a formal construction of an **At**-derivation $\Delta_{\mathrm{add}}(A, Y, Z)$ of $\mathcal{D}(A \cup \{Y \oplus Z\})$ from $\mathcal{D}(A)$, for parity constraints Y, Z with intersecting variable sets. As in the sketch

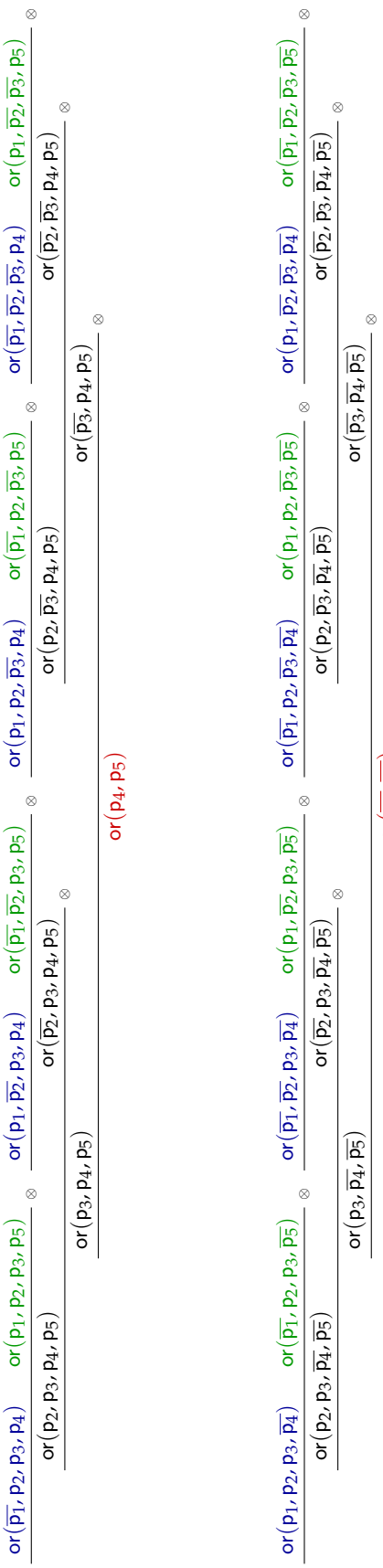


Figure 6.3: Resolution trees deriving the clauses in $\mathcal{D}(Y \oplus Z) = \{\mathsf{or}(p_4, p_5), \mathsf{or}(\neg p_4, \neg p_5)\}$, shown in color red, from clauses in $\mathcal{D}(Y)$, shown in blue, or in $\mathcal{D}(Z)$, shown in green.

above, enumerate $\mathrm{vars}(Y) \cap \mathrm{vars}(Z)$ as $\{a_1, \ldots, a_n\}$, and denote $Y \oplus Z$ by X. We define sets of variables U_j and CNF formulae H_j for every $0 \leq j \leq n$:

$$\begin{aligned} U_j &= (\mathrm{vars}(Y) \cup \mathrm{vars}(Z)) \setminus \{a_1, \ldots, a_j\} \\ H_0 &= \mathcal{D}(\{Y, Z\}) \\ H_j &= \{\mathrm{nand}(V) \vee \mathsf{or}(U_j \setminus V) : V \subseteq U_j \text{ and } |V \cap \mathrm{vars}(X)| \approx \mathrm{pol}(X) + 1\} \qquad \text{for } 1 \leq j \leq n \end{aligned}$$

Our construction works as shown in Figure 6.4. First, for every $1 \leq j \leq n$ we give **At**-derivations α_j of H_j from H_{j-1} defined by:

$$\alpha_j = [H_{j-1}\ ,\ H_{j-1} \cup H_j\ ,\ H_j]$$

These derivations are then concatenated together to obtain an **At**-derivation $\alpha = \alpha_1 : \ldots : \alpha_n$ of H_n from H_0. The CNF formula H_0 is a subset of the direct encoding of A. Furthermore, as we show in Lemma 6.6, the CNF formula H_n is precisely the direct encoding of $X = Y \oplus Z$. Therefore, the fusion $\mathcal{D}(A) \sqcup \alpha$ is an **At**-derivation of $\mathcal{D}(A \cup \{Y \oplus Z\})$ from $\mathcal{D}(A)$; this will be our translation $\Delta_{\mathrm{add}}(A, Y, Z)$.

Since degenerate cases were given translations in Proposition 6.1, we assume that parity constraints Y and Z satisfy the following conditions:

i) $\mathrm{vars}(Y)$ and $\mathrm{vars}(Z)$ are not disjoint: parity constraints Y and Z contain some variable in common.

ii) $Y \oplus Z \notin A$: the parity constraint addition inference is not redundant.

iii) $Y \neq Z$: the parity constraint addition inference is not the empty parity constraint $\mathsf{or}(\emptyset)$.

Observe that the CNF formulae H_i only are well-defined under condition *(i)*. We do not write these conditions along the results below. Let us first compute cardinalities of the H_i, which will be helpful when finding length bounds for the translation.

Lemma 6.3 (Size of transitional CNF formulae):
For every $1 \leq i \leq n$*, we have* $|H_i| = 2^{\omega(Y,Z)-i}$ *if* $Y \oplus Z = \mathsf{par}(\top)$*; and* $|H_i| = 2^{\omega(Y,Z)-i-1}$ *otherwise.*

Proof. To show the claim, we define a bijection between H_i and the cartesian product $P \times Q$, where the sets of variables P and Q are given by:

$$P = \{V \subseteq \mathrm{vars}(Y \oplus Z) : |V| \approx \mathrm{pol}(Y \oplus Z) + 1\} \qquad Q = \mathcal{P}(\{a_{i+1}, \ldots, a_n\})$$

Let us define two mappings Φ and Ψ from H_i into $P \times Q$ and viceversa:

$$\begin{aligned} \Phi\colon\ & P \times Q \longrightarrow H_i \\ & \langle V, W \rangle \ \mapsto\ \Phi(V, W) = \mathrm{nand}(V \cup W) \vee \mathsf{or}(U_i \setminus (V \cup W)) \end{aligned}$$

$$\begin{aligned} \Psi\colon\ & H_i \longrightarrow P \times Q \\ & C \ \mapsto\ \Psi(C) = \langle \{b \in \mathrm{vars}(Y \oplus Z) : \neg b \in \mathrm{lits}(C)\}\ ,\ \{a_j : i+1 \leq j \leq n \text{ and } \neg a_j \in \mathrm{lits}(C)\} \rangle \end{aligned}$$

Observe that $\Phi(V, W)$ constructs a clause with negative literals from variables in $V \cup W$, and positive literals from the rest of variables in $U_i = \mathrm{vars}(Y \oplus Z) \cup \{a_{i+1}, \ldots, a_n\}$. On the other hand, $\Psi(C)$ finds the underlying variables to negative literals in C, and segregates them depending on whether they lie in $\mathrm{vars}(Y) \triangle \mathrm{vars}(Z)$ or in $\mathrm{vars}(Y) \cap \mathrm{vars}(Z)$, which are disjoint sets. From these considerations, it can be easily checked that $\Phi \circ \Psi$ is the identity mapping over H_i, and $\Psi \circ \Phi$ is the identity mapping over $P \times Q$. Therefore, Φ and Ψ are mutually inverse bijections.

In particular, we conclude that $|H_i| = |P \times Q| = |P| \cdot |Q|$. Q is just the power set of $\{a_{i+1}, \ldots, a_n\}$, hence $|Q| = 2^{n-i}$. We now distinguish two cases:

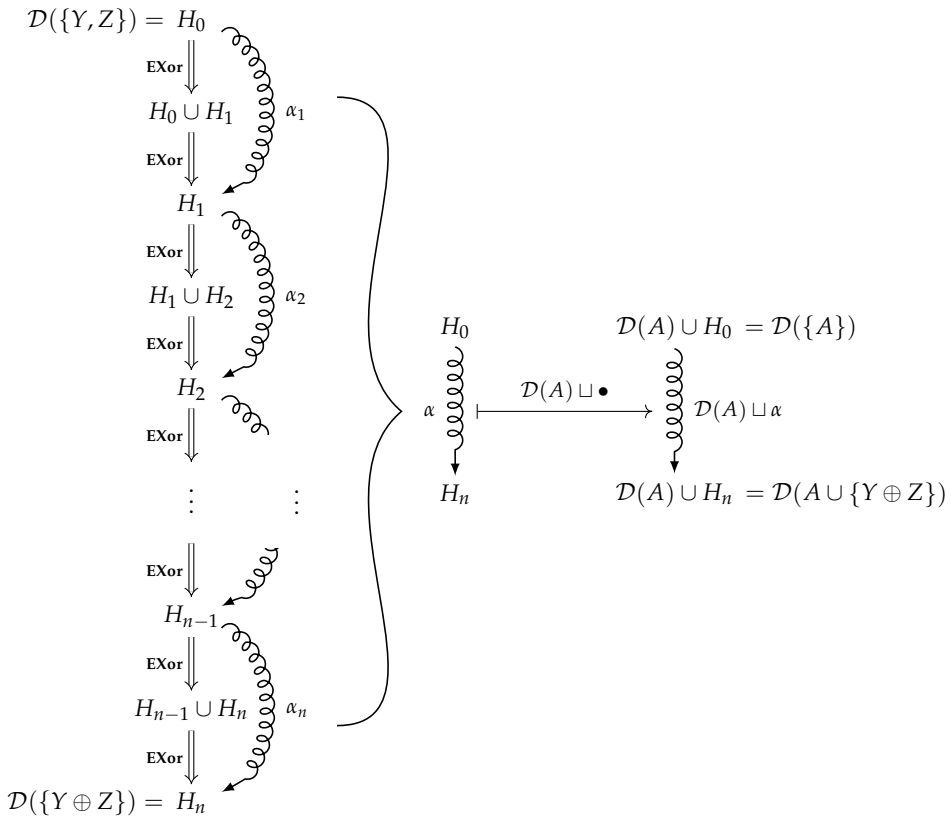


Figure 6.4: Direct translation of (non-degenerate) parity constraint addition inferences. The **Rat**-derivations $\alpha_i = [H_{i-1}, H_{i-1} \cup H_i, H_i]$ are concatenated to obtain a **Rat**-derivation α of $H_n = \mathcal{D}(Y \oplus Z)$ from $H_0 = \mathcal{D}(\{Y,Z\})$. Fusion of α with the CNF formula $\mathcal{D}(A)$ yields the direct translation $\Delta_{\text{add}}(A,Y,Z)$ of the parity constraint addition inference of Y and Z over A. As usual, $\Rightarrow$ arrows represent inferences, "coil" arrows represent **Rat**-derivations and $\mapsto$ arrows represent the application of a mapping, where the argument is denoted by $\bullet$. Furthermore, braces denote concatenation of derivations.

- If $Y \oplus Z = \mathsf{par}(\top)$, then $\text{vars}(Y \oplus Z) = \emptyset$ and $\text{pol}(Y \oplus Z) + 1 \approx 0$. Hence, by Proposition 2.2, we have $|P| = 1$. Observe that in this case $\text{vars}(Y) = \text{vars}(Z)$, so we have $n = \omega(Y,Z)$ and therefore $|H_i| = |Q| = 2^{\omega(Y,Z)-i}$.
- If $Y \oplus Z \neq \mathsf{par}(\top)$, then by the assumption that $Y \neq Z$ we know that $Y \oplus Z \neq \mathsf{par}(\emptyset)$. Hence, $\text{vars}(Y \oplus Z)$ is nonempty and then Proposition 2.2 yields $|P| = 2^{|\text{vars}(Y \oplus Z)|-1} = 2^{\delta(Y,Z)-1}$ and therefore $|H_i| = 2^{\delta(Y,Z)+n-i-1}$. Finally, note that

$$\delta(Y,Z) + n = |\text{vars}(Y) \mathbin{\triangle} \text{vars}(Z)| + |\text{vars}(Y) \cap \text{vars}(Z)| = |\text{vars}(Y) \cup \text{vars}(Z)| = \omega(Y,Z)$$

  so we conclude $|H_i| = 2^{\omega(Y,Z)-i-1}$. □

Henceforth, we show that $\mathcal{D}(A) \sqcup \alpha$ is an **At**-derivation of $\mathcal{D}(A \cup \{X\})$ from $\mathcal{D}(A)$. Lemmata 6.4 and 6.5 show that every string α_i as defined above is an **At**-derivation. Lemma 6.6 proves that the concatenation $\alpha = \alpha_1 : \ldots : \alpha_n$ derives $\mathcal{D}(X)$ from $\mathcal{D}(A)$ by showing H_n to be the direct translation of X. Finally, in Theorem 6.7 the claim above for $\mathcal{D}(A) \sqcup \alpha$ is showed.

Lemma 6.4 (Correctness of transitional fragments, base case):
$\alpha_1 = [H_0\ ,\ H_0 \cup H_1\ ,\ H_1]$ *is an* **At**-*derivation of* H_1 *from* H_0.

Proof. The second inference $H_0 \cup H_1 \Rightarrow_{\mathbf{At}} H_1$ holds straightforward by clause deletion. We now focus on showing that the first inference $H_0 \Rightarrow_{\mathbf{At}} H_0 \cup H_1$ holds as well by asymmetric tautology inference. That is, we show that every clause $C \in H_1$ is an asymmetric tautology in H_0; then, $H_1 \subseteq H_0^{\mathsf{AT}}$ holds and the inference follows.

The clause C is of the form $\mathrm{nand}(V) \vee \mathsf{or}(U_1 \setminus V)$ for a set of variables

$$V \subseteq U_1 = (\mathrm{vars}(Y) \cup \mathrm{vars}(Z) \setminus \{a_1\}) \quad \text{such that} \quad |V \cap \mathrm{vars}(X)| \approx \mathrm{pol}(X) + 1$$

We now show that the quantities $s = |V \cap \mathrm{vars}(Y)| + |V \cap \mathrm{vars}(Z)|$ and $|V \cap \mathrm{vars}(X)|$ are congruent modulo 2. Let us partition the sets above as follows:

$$V \cap \mathrm{vars}(Y) = \Big(V \cap (\mathrm{vars}(Y) \setminus \mathrm{vars}(Z))\Big) \cup \Big(V \cap \mathrm{vars}(Y) \cap \mathrm{vars}(Z)\Big)$$
$$V \cap \mathrm{vars}(Z) = \Big(V \cap (\mathrm{vars}(Z) \setminus \mathrm{vars}(Y))\Big) \cup \Big(V \cap \mathrm{vars}(Y) \cap \mathrm{vars}(Z)\Big)$$

The sets in the right are disjoint, so we obtain the following equality:

$$s = |V \cap (\mathrm{vars}(Y) \setminus \mathrm{vars}(Z))| + 2|V \cap \mathrm{vars}(Y) \cap \mathrm{vars}(Z)| + |V \cap (\mathrm{vars}(Z) \setminus \mathrm{vars}(Y))|$$

We are only interested on computing the parity of s, so we can simplify this to:

$$s \approx |V \cap (\mathrm{vars}(Y) \setminus \mathrm{vars}(Z))| + |V \cap (\mathrm{vars}(Z) \setminus \mathrm{vars}(Y))|$$

The sets above are again disjoint, and by simple set identities we obtain that s is congruent modulo 2 to:

$$\Big|V \cap \Big((\mathrm{vars}(Y) \setminus \mathrm{vars}(Z)) \cup (\mathrm{vars}(Z) \setminus \mathrm{vars}(Y))\Big)\Big| = \Big|V \cap (\mathrm{vars}(Y) \triangle \mathrm{vars}(Z))\Big| = \Big|V \cap \mathrm{vars}(X)\Big|$$

as we wanted. From the parity condition on V, the following non-congruence follows:

$$|V \cap \mathrm{vars}(Y)| + |V \cap \mathrm{vars}(Z)| \approx |V \cap \mathrm{vars}(X)| \not\approx \mathrm{pol}(X) \approx \mathrm{pol}(Y) + \mathrm{pol}(Z)$$

By Proposition 2.1, exactly one of the following holds:

$$|V \cap \mathrm{vars}(Y)| \approx \mathrm{pol}(Y) \qquad |V \cap \mathrm{vars}(Z)| \approx \mathrm{pol}(Z) \tag{6.1}$$

Without loss of generality, we assume that the congruence in the left is falsified and the one in the right is satisfied; the opposite case is shown by a symmetric argument. Observe now that $a_1 \notin V$. Define the following sets and clauses:

$$V_Y = V \cap \mathrm{vars}(Y) \qquad V_Z = (V \cap \mathrm{vars}(Z)) \cup \{a_1\}$$
$$D_Y = \mathrm{nand}(V_Y) \vee \mathsf{or}(\mathrm{vars}(Y) \setminus V_Y) \qquad D_Z = \mathrm{nand}(V_Z) \vee \mathsf{or}(\mathrm{vars}(Z) \setminus V_Z)$$

By our choice on the congruences (6.1), the following hold, showing that $D_Y \in \mathcal{D}(Y)$ and $D_Z \in \mathcal{D}(Z)$:

$$|V_Y| \approx \mathrm{pol}(Y) + 1 \qquad |V_Z| \approx \mathrm{pol}(Z) + 1$$

We now show that D_Y is resolvable with D_Z upon a_1 with resolvent $C = \mathrm{nand}(V) \vee \mathsf{or}(U_1 \setminus V)$.

- *The literal a_1 occurs in D_Y, and the literal $\neg a_1$ occurs in D_Z* follows from $a_1 \notin V_Y$ and $a_1 \in V_Z$.
- *For every literal l in D_Y other than a_1, its complement $\bar{l}$ does not occur in D_Z* if l was a literal with this property, then its underlying variable $\mathrm{var}(l)$ would be in $\mathrm{vars}(Y) \cap \mathrm{vars}(Z)$. If $\mathrm{var}(l) \in V$, then this

means that both $\text{var}(l) \in V_Y$ and $\text{var}(l) \in V_Z$ hold, and therefore $\text{var}(l)$ occurs only as a positive literal in both D_Y and D_Z. Otherwise, $\text{var}(l) \notin V$, which implies that $\text{var}(l) \notin V_Y$ and $\text{var}(l) \notin V_Z$ hold, and therefore $\text{var}(l)$ occurs only as a negative literal in both D_Y and D_Z. Both conclusions are a contradiction with the claim.

- *For every literal l in D_Z other than $\neg a_1$, its complement $\bar{l}$ does not occur in D_Y* it is shown analogously to the previous property.
- *$D_Y \otimes_{a_1} D_Z = C$ holds* observe that positive literals in the resolvent have underlying variables:

$$((\text{vars}(Y) \setminus V_Y) \cup (\text{vars}(Z) \setminus V_Z)) \setminus \{a_1\} = ((\text{vars}(Y) \setminus \{a_1\}) \setminus V) \cup ((\text{vars}(Z) \setminus \{a_1\}) \setminus V) = \\ = ((\text{vars}(Y) \cup \text{vars}(Z)) \setminus \{a_1\}) \setminus V = U_1 \setminus V$$

which are precisely the underlying variables to positive literals in C. Similarly, negative literals in the resolvent have underlying variables

$$(V_Z \cup V_Y) \setminus \{a_1\} = (V \cap \text{vars}(Y)) \cup (V \cap \text{vars}(Z)) = V \cap (\text{vars}(Y) \cup \text{vars}(Z)) = V$$

which are the underlying variables to negative literals in C. Therefore, the resolvent contains precisely the same literals as C.

We have shown that C is a resolvent of clauses in $\mathcal{D}(\{Y, Z\}) = H_0$, thus an asymmetric tautology in H_0 as we wanted to show. Then, α is a **At**-derivation as claimed. □

Lemma 6.5 (Correctness of transitional fragments, induction step):
For every $2 \leq i \leq n$, the string $\alpha_i = [H_{i-1}\ ,\ H_{i-1} \cup H_i\ ,\ H_i]$ is an **At***-derivation of H_i from H_{i-1}.*

Proof. As in the proof of Lemma 6.4, the second inference $H_{i-1} \cup H_i \Rightarrow_{\mathbf{At}} H_i$ holds by clause deletion, which is straightforward, and $H_{i-1} \Rightarrow_{\mathbf{At}} H_{i-1} \cup H_i$ holds by asymmetric tautology inference. To show the latter, we prove every clause $C \in H_i$ to be an asymmetric tautology in H_{i-1}. In particular, we prove that the clauses $D^+ = C \vee \mathsf{or}(a_i)$ and $D^- = C \vee \mathsf{or}(\neg a_i)$ occur in H_{i-1}. Hence, C is a resolvent of two clauses from H_{i-1}, so by Proposition 3.16 it is an asymmetric tautology in H_{i-1}.

The clause C is of the form $\text{nand}(V) \vee \mathsf{or}(U_i \setminus V)$ for a set of variables:

$$V \subseteq U_i = (\text{vars}(Y) \cup \text{vars}(Z) \setminus \{a_1, \ldots, a_i\}) \quad \text{such that} \quad |V \cap \text{vars}(X)| \approx \text{pol}(X) + 1$$

Consider sets of variables $V^+ = V$ and $V^- = V \cup \{a_i\}$. It is straightforward that both are subsets of $U_{i-1} = U_i \cup \{a_i\}$. Furthermore, since $a_i \in \text{vars}(Y) \cap \text{vars}(Z)$, it follows that $a_i \notin \text{vars}(X)$. Hence,

$$|V^+ \cap \text{vars}(X)| = |V^- \cap \text{vars}(X)| = |V \cap \text{vars}(X)| \approx \text{pol}(X) + 1$$

and consequently the clauses

$$D^+ = \text{nand}(V^+) \vee \mathsf{or}(U_{i-1} \setminus V^+) \qquad D^- = \text{nand}(V^-) \vee \mathsf{or}(U_{i-1} \setminus V^-)$$

belong to H_{i-1}. Since $a_i \notin \text{vars}(C)$, we obtain that D^+ and D^- are resolvable upon a_i with resolvent:

$$D^+ \otimes_{a_i} D^- = (C \vee \mathsf{or}(a_i)) \otimes_{a_i} (C \vee \mathsf{or}(\neg a_i)) = C$$

Therefore, C is an asymmetric tautology in H_{i-1}, as we wanted. This shows that α_i is an **At**-derivation. □

Lemma 6.6 (Transitional CNF formulae derive the encoded addition):
$H_n = \mathcal{D}(Y \oplus Z)$.

Proof. The definition of U_n and H_n are given by:

$$U_n = (\mathrm{vars}(Y) \cup \mathrm{vars}(Z)) \setminus \{a_1, \ldots, a_n\}$$
$$H_n = \{\mathrm{nand}(V) \vee \mathsf{or}(U_n \setminus V) : V \subseteq U_n \text{ and } |V \cap \mathrm{vars}(X)| \approx \mathrm{pol}(X) + 1\}$$

Since $\{a_1, \ldots, a_n\} = \mathrm{vars}(Y) \cap \mathrm{vars}(Z)$, we obtain $U_n = \mathrm{vars}(Y) \triangle \mathrm{vars}(Z) = \mathrm{vars}(Y \oplus Z) = \mathrm{vars}(X)$, and consequently the definition of H_n coincides with that of $\mathcal{D}(X)$. □

Theorem 6.7 (Correctness of direct translations of addition inferences, regular case):
The string $\mathcal{D}(A) \sqcup \alpha$ is an **At***-derivation of the direct encoding $\mathcal{D}(A \cup \{Y \oplus Z\})$ from the direct encoding $\mathcal{D}(A)$.*

Proof. From Lemmas 6.4 and 6.5 we conclude that the concatenation $\alpha = \alpha_1 : \ldots : \alpha_n$ is an **At**-derivation of H_n from $\mathcal{D}(\{Y, Z\})$. Lemma 6.6 shows that $H_n = \mathcal{D}(Y \oplus Z)$, and then α is a **Rat**-derivation of $\mathcal{D}(Y \oplus Z)$ from $\mathcal{D}(\{Y, Z\})$ sharp in $\emptyset$. By Proposition 3.35, $\mathcal{D}(A) \sqcup \alpha$ is a **Rat**-derivation of $\mathcal{D}(A) \cup \mathcal{D}(Y \oplus Z) = \mathcal{D}(A \cup \{Y \oplus Z\})$ from $\mathcal{D}(A) \cup \mathcal{D}(\{Y, Z\}) = \mathcal{D}(A)$ sharp in $\emptyset$ (therefore, an **At**-derivation). □

We now give a definition of the translation of an addition inference. Observe that due to degenerate cases we need to make a case distinction. Furthermore, the definition is not univocal, in the sense that our construction depends on the chosen enumeration $\{a_1, \ldots, a_n\}$. This is irrelevant for our goals, so in an abuse of notation we assign a symbol to some derivation constructed as proposed here.

Definition 6.8 (Direct translation of addition inferences):
Let A be an affine formula and $Y, Z \in A$ be two parity constraints. We define the direct translation of the addition inference *of Y and Z over A as an* **At***-derivation $\Delta_{\mathrm{add}}(A, Y, Z)$ constructed as follows:*

i) If $Y \oplus Z \in A$ or $Y = Z$, then
$\Delta_{\mathrm{add}}(A, Y, Z) = [\mathcal{D}(A)]$.

ii) If $Y \oplus Z \notin A$, $Y \neq Z$ and $\mathrm{vars}(Y)$ is disjoint from $\mathrm{vars}(Z)$, then
$\Delta_{\mathrm{add}}(A, Y, Z) = [\mathcal{D}(A)\ ,\ \mathcal{D}(A \cup \{Y \oplus Z\})]$.

iii) If $Y \oplus Z \notin A$, $Y \neq Z$ and $\mathrm{vars}(Y) \cap \mathrm{vars}(Z)$ is nonempty, then
$\Delta_{\mathrm{add}}(A, Y, Z)$ is the derivation $\Delta_{\mathrm{add}}(A, \sqcup, \alpha)$ as defined before in this section.

By Proposition 6.1 and Theorem 6.7, $\Delta_{\mathrm{add}}(A, Y, Z)$ is an **At***-derivation of $\mathcal{D}(A \cup \{Y \oplus Z\})$ from $\mathcal{D}(A)$.*

Before the construction of a full direct translation using the translation just presented is introduced, we give some length bounds for the construction above. As can be observed from the proof, the obtained bounds are "almost exact", in the sense that the only reason why they may be strictly greater than the actual length is that some derived clauses might be already in the set of premises by coincidence. This allows to compute the length (up to "coincidence") of the translation in advance.

Proposition 6.9 (Lengths of direct translations of addition inferences):
Let A be an affine formula and $Y, Z \in A$ be two parity constraints. The length of the direct translation of the addition inference of Y and Z over A is bounded as follows:

i) If $Y \oplus Z \in A$ or $Y = Z$, then
$\ell(\Delta_{\mathrm{add}}(A, Y, Z)) = 0$.

ii) If $Y \oplus Z \notin A$, and $Y \neq Z$, and $\mathrm{vars}(Y)$ is disjoint from $\mathrm{vars}(Z)$, and $Y \oplus Z \neq \mathsf{par}(\top)$, then
$\ell(\Delta_{\mathrm{add}}(A, Y, Z)) \leq 2^{\delta(Y,Z)-1}$.

iii) If $Y \oplus Z \notin A$, and $Y \neq Z$, and $\mathrm{vars}(Y)$ is disjoint from $\mathrm{vars}(Z)$, and $Y \oplus Z = \mathsf{par}(\top)$, then
$\ell(\Delta_{\mathrm{add}}(A, Y, Z)) \leq 1$.

iv) *If* $Y \oplus Z \notin A$, *and* $Y \neq Z$, *and* $\mathrm{vars}(Y) \cap \mathrm{vars}(Z)$ *is nonempty, and* $Y \oplus Z \neq \mathsf{par}(\top)$, *then* $\ell(\Delta_{\mathrm{add}}(A, Y, Z)) \leq 2^{\omega(Y,Z)} - \frac{3}{2} \cdot 2^{\delta(Y,Z)}$.

v) *If* $Y \oplus Z \notin A$, *and* $Y \neq Z$, *and* $\mathrm{vars}(Y) \cap \mathrm{vars}(Z)$ *is nonempty, and* $Y \oplus Z = \mathsf{par}(\top)$, *then* $\ell(\Delta_{\mathrm{add}}(A, Y, Z)) \leq 2^{\omega(Y,Z)+1} - 3$.

Proof.

i) In this case, we have $\Delta_{\mathrm{add}}(A, Y, Z) = [\mathcal{D}(A)]$, which has length 0.

ii, iii) The translation in these cases is given by $[\mathcal{D}(A)\ ,\ \mathcal{D}(A \cup \{Y \oplus Z\})]$, whose length is bounded by $s = |\mathcal{D}(\{Y \oplus Z\})|$. A computation of this quantity is given in Proposition 5.4. If $Y \oplus Z = \mathsf{par}(\top)$, then we obtain $s = 1$. Otherwise, $Y \oplus Z \neq \mathsf{par}(\top)$, and since $Y \neq Z$ then $Y \oplus Z \neq \mathsf{par}(\emptyset)$. Hence, $s = 2^{|Y \oplus Z|-1} = 2^{\delta(Y,Z)-1}$.

iv, v) For these two cases, the translation is given by $\mathcal{D}(A) \sqcup (\alpha_1 : \ldots : \alpha_n)$, where the α_i are given by $[H_{i-1}\ ,\ H_{i-1} \cup H_i\ ,\ H_i]$ for every $1 \leq i \leq n$. Since $H_0 \subseteq \mathcal{D}(A)$, the translation can be decomposed as:

$$\Delta_{\mathrm{add}}(A, Y, Z) = [\mathcal{D}(A)\ ,\ \mathcal{D}(A) \cup H_1\ ,\ \mathcal{D}(A) \cup H_1] : (\mathcal{D}(A) \sqcup (\alpha_2 : \ldots : \alpha_n))$$

Observe that every α_i with $i > 2$ has length bounded by $|H_{i-1}| + |H_i|$, and hence we obtain by Proposition 3.41:

$$\ell(\Delta_{\mathrm{add}}(A, Y, Z)) = |H_1| + \sum_{i=2}^{n} (|H_{i-1}| + |H_i|) = 2 \sum_{i=1}^{n-1} |H_i| + |H_n| \tag{6.2}$$

Computations for $|H_i|$ are provided in Lemma 6.3. We distinguish the two cases in the claim:

iv) If $Y \oplus Z \neq \mathsf{par}(\top)$ we obtain from Lemma 6.3 that $|H_i| = 2^{\omega(Y,Z)-i-1}$ for all $1 \leq i \leq n$. Observe that, for $i = n$, the equality $\omega(Y, Z) = \delta(Y, Z) + |\mathrm{vars}(Y) \cap \mathrm{vars}(Z)|$ holds, and this implies $|H_n| = 2^{\delta(Y,Z)-1}$. This yields the right hand summand in (6.2). We now simplify the sum. Using Lemma 6.3 we obtain:

$$2 \sum_{i=1}^{n-1} |H_i| = \sum_{i=1}^{n-1} 2^{\omega(Y,Z)-i}$$

We can perform the change of variables $j = \omega(Y, Z) - i$. Since $\omega(Y, Z) - n = \delta(Y, Z)$, we obtain

$$2 \sum_{i=1}^{n-1} |H_i| = \sum_{j=\delta(Y,Z)+1}^{\omega(Y,Z)-1} 2^i$$

which is a partial sum of a geometric series. By Proposition 2.3, this simplifies to:

$$2 \sum_{i=1}^{n-1} |H_i| = 2^{\omega(Y,Z)} - 2^{\delta(Y,Z)+1}$$

With these simplifications, the bound is:

$$\ell(\mathcal{D}(A) \sqcup \alpha) \leq 2^{\omega(Y,Z)} - 2^{\delta(Y,Z)+1} + 2^{\delta(Y,Z)-1} = 2^{\omega(Y,Z)} - \frac{3}{2} \cdot 2^{\delta(Y,Z)}$$

v) If $Y \oplus Z = \mathsf{par}(\top)$ we obtain $|H_i| = 2^{\omega(Y,Z)-i}$ for all $1 \leq i \leq n$ by Lemma 6.3. An analogous computation to the previous case using that now $\omega(Y, Z) = n$ gives the bound:

$$\ell(\mathcal{D}(A) \sqcup \alpha) \leq 2 \sum_{i=1}^{n-1} 2^{\omega(Y,Z)-i} + 1 = \sum_{i=2}^{\omega(Y,Z)} 2^i + 1 = 2^{\omega(Y,Z)+1} - 3 \qquad \square$$

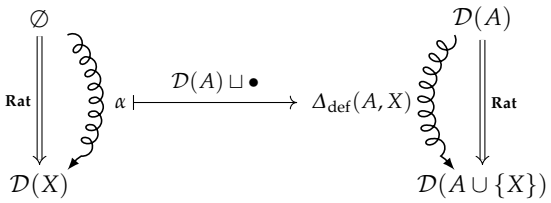


Figure 6.5: Direct translation of XOR definition introduction inferences. The **Rat**-derivation $\alpha = [\emptyset, \mathcal{D}(X)]$ is fused with the CNF fomula $\mathcal{D}(A)$ to obtain the direct translation $\Delta_{\text{def}}(A, X)$ of the inference $A \Rightarrow_{\mathbf{EXor}} A \cup \{X\}$. As usual, $\Rightarrow$ arrows represent inferences, "coil" arrows represent **Rat**-derivations and $\mapsto$ arrows represent the application of a mapping, where the argument is denoted by $\bullet$.

6.2 Construction of the full direct translation

This section is devoted to the construction of the full direct translation of an **EXor**-derivation. As shown in Figure 6.1, the construction relies upon subtranslations for every inference along the original **EXor**-derivation. So far, we have only obtained translations for parity constraint addition inferences; we now provide translations for XOR definition introduction and parity constraint deletion inferences. Afterwards, the full direct translation is presented.

Direct translation of XOR definition introduction inferences. We construct a **Rat**-derivation $\Delta_{\text{def}}(A, X)$ of the direct encoding $\mathcal{D}(A \cup \{X\})$ of an affine formula $A \cup \{X\}$ from $\mathcal{D}(A)$ for the case when the parity constraint X is a XOR definition in A. Observe that, since XOR definition introduction is only cosatisfiability-sound, no **At**-derivation with the requirements above exist. In this case, the derivation is rather simple: the whole direct encoding $\mathcal{D}(X)$ can be introduced in a single inference as an unresolvable set of resolution asymmetric tautologies. We provide an **Rat**-derivation α of $\mathcal{D}(X)$ from the empty CNF formula, which is then fused with the CNF formula $\mathcal{D}(A)$, as shown in Figure 6.5. Whenever an affine formula A does not contain a variable a occurring in X, then $\mathcal{D}(A)$ can be fused with α to obtain an **Rat**-derivation of $\mathcal{D}(A \cup \{X\})$ from $\mathcal{D}(A)$; we show this by showing that $\{a\}$ is an appropiate sharpness set.

Let us consider the string of CNF formulae $\alpha = [\emptyset\,,\ \mathcal{D}(X)]$. We now show that the only inference $\emptyset \Rightarrow_{\mathbf{Rat}} \mathcal{D}(X)$ holds. Since the variable a does not occur in the empty CNF formula, then $\mathcal{D}(X)$ is a set of resolution asymmetric tautologies in $\emptyset$. It only remains to see that the direct encoding of any parity constraint is unresolvable.

Lemma 6.10 (Encoded XOR definitions are unresolvable):
Let X be a parity constraint. Then, $\mathcal{D}(X)$ is unresolvable.

Proof. We need to show that, given two clauses $C, D \in \mathcal{D}(X)$, the clause C is not resolvable with D. We reason by contradiction. Let $C, D \in \mathcal{D}(X)$, and assume that C is resolvable with D upon a literal l. Observe that $\text{vars}(C) = \text{vars}(D) = \text{vars}(X)$. Let $k \in \text{lits}(C)$ be other than l. Then, either k or $\overline{k}$ occurs in D. However, $\overline{k}$ cannot occur in D, because that would violate the condition that C is resolvable with D. Therefore, every literal in C other than l occurs in D as well. By a symmetric argument, it also holds that every literal in D other than $\overline{l}$ occurs in C. Therefore, $\text{lits}(C) \setminus \{l\} = \text{lits}(D) \setminus \{\overline{l}\}$. Define the sets:

$$V_C = \{a \in \text{vars}(X) : \neg a \in \text{lits}(C)\} \qquad V_D = \{a \in \text{vars}(X) : \neg a \in \text{lits}(D)\}$$

Then, we can express clauses C and D as

$$C = \text{nand}(V_C) \vee \text{or}(\text{vars}(X) \setminus V_C) \qquad D = \text{nand}(V_D) \vee \text{or}(\text{vars}(X) \setminus V_D)$$

and by the definition of direct encoding, this implies that $|V_C| \approx \text{pol}(X) + 1 \approx |V_D|$. However, the arguments above imply that $\text{var}(l) \notin V_C$ and $V_D = V_C \cup \{\text{var}(l)\}$. Therefore, $|V_C| \approx |V_D| \approx |V_C| + 1$, and substracting $|V_C|$, the congruence $0 \approx 1$ follows, which is a contradiction. By reduction to absurd, C is not resolvable with D. □

Definition 6.11 (Direct translation of a XOR definition introduction inference):
Let A be an affine formula and X be a XOR definition in A. We define the direct translation of the XOR definition inference *of X over A as the string of CNF formulae $\Delta_{\text{def}}(A, X) = [\mathcal{D}(A)\ ,\ \mathcal{D}(A \cup \{X\})]$.*

Theorem 6.12 (Correctness of direct translations of XOR definition introduction inferences):
Let A be an affine formula and X be a XOR definition in A upon a variable a. Then, the direct translation $\Delta_{\text{def}}(A, X)$ of the XOR definition inference of X over A is a **Rat***-derivation of $\mathcal{D}(A \cup \{X\})$ from $\mathcal{D}(A)$ sharp in $\{a\}$. Furthermore, the length of $\Delta_{\text{def}}(A, X)$ is bounded by $2^{|X|-1}$.*

Proof. Since X is a XOR definition in A upon a, we have $a \in \text{vars}(X) \setminus \text{vars}(A)$. Consider the string of CNF formulae $\alpha = [\emptyset\ ,\ \mathcal{D}(X)]$, and let C be an arbitrary clause in $\mathcal{D}(X)$. By the definition of direct encoding, a literal l with underlying variable $\text{var}(l) = a$ occurs in C. $a \notin \text{vars}(A)$ implies that there is no clause in $\emptyset$ resolvable with C upon l, so C is a resolution asymmetric tautology in $\emptyset$ upon a. Then, $\mathcal{D}(X)$ is an unresolvable (see Lemma 6.10) set of resolution asymmetric tautologies in $\emptyset$ upon a, so α is a **Rat**-derivation of $\mathcal{D}(X)$ from $\emptyset$ sharp in $\{a\}$.

Now, observe that $\Delta_{\text{def}}(A, X) = \mathcal{D}(A) \sqcup \alpha$, and that since a does not occur in A, then $a \notin \mathcal{D}(A)$. By Proposition 3.35, this shows that $\Delta_{\text{def}}(A, X)$ is a **Rat**-derivation of $\mathcal{D}(A) \cup \mathcal{D}(X) = \mathcal{D}(A \cup \{X\})$ from $\mathcal{D}(A)$ sharp in $\{a\}$.

To obtain the bound, note that X contains at least variable a, so $|X| > 1$. Therefore:

$$\ell(\Delta_{\text{def}}(A, X)) \leq \ell(\alpha) = |\emptyset \mathbin{\triangle} \mathcal{D}(X)| = |\mathcal{D}(X)| = 2^{|X|-1}$$ □

Direct translation of parity constraint deletion inferences. A **At**-derivation of $\mathcal{D}(B)$ from $\mathcal{D}(A)$ for the case when $B \subseteq A$ can be readily constructed, since this condition implies $\mathcal{D}(B) \subseteq \mathcal{D}(A)$ and hence $[\mathcal{D}(A)\ ,\ \mathcal{D}(B)]$ is an **At**-derivation as required.

Definition 6.13 (Direct translations of parity constraint deletion inferences):
Let A and B be affine formulae with $B \subseteq A$. We define the direct translation $\Delta_{\text{del}}(A, B)$ of the parity constraint deletion inference *of B over A as the string of CNF formulae $[\mathcal{D}(A)\ ,\ \mathcal{D}(B)]$.*

Theorem 6.14 (Correctness of direct translations of parity constraint deletion inferences):
Let A and B be affine formulae with $B \subseteq A$. Then, the direct translation $\Delta_{\text{del}}(A, B)$ of the parity constraint deletion inference of B over A is an **At***-derivation of $\mathcal{D}(B)$ from $\mathcal{D}(A)$. Furthermore, if we define*

$$s_X = 2^{|X|-1}, \quad \text{if } |X| > 1 \qquad s_X = 0, \quad \text{if } X = \mathsf{par}(\emptyset) \qquad s_X = 1, \quad \text{if } X = \mathsf{par}(\top)$$

for every parity constraint $X \in A \setminus B$, then the length of $\Delta_{\text{del}}(A, X)$ is bounded by $\sum_{X \in A} s_X$.

Proof. $A \Rightarrow_{\mathbf{At}} B$ follows from $B \subseteq A$, and then $\Delta_{\text{del}}(A, B)$ is a derivation as indicated. The length bound follows from observing that $s_X = |\mathcal{D}(X)|$ for every $X \in A \setminus B$. Hence:

$$\ell(\Delta_{\text{del}}(A, B)) = |\mathcal{D}(A) \mathbin{\triangle} \mathcal{D}(B)| = |\mathcal{D}(A) \setminus \mathcal{D}(B)| = |\mathcal{D}(A \setminus B)| = \sum_{X \in A \setminus B} |\mathcal{D}(X)| = \sum_{X \in A \setminus B} s_X$$ □

Full direct translation of EXor-derivations. We can finally define full direct translations. The definition relies on iteratively concatenating derivations generated earlier in this chapter for every inference along an **EXor**-derivation, as shown in Figure 6.1.

Definition 6.15 (Direct translations of EXor-derivations):
Let $\varphi = [A_0, \ldots, A_n]$ be an **EXor***-derivation of A_n from A_0 sharp in V. For every $1 \leq i \leq n$, consider a string of CNF formulae α_i as follows:*

i) If $A_i \subseteq A_{i-1}$, then let $\alpha_i = \Delta_{\mathrm{del}}(A_{i-1}, A_i)$.

ii) If $A_i = A_{i-1} \cup \{Y \oplus Z\}$ for some parity constraints $Y, Z \in A_{i-1}$, then let $\alpha_i = \Delta_{\mathrm{add}}(A_{i-1}, X, Y)$.

iii) If $A_i = A_{i-1} \cup \{X\}$ for some XOR definition X in A_{i-1}, then let $\alpha_i = \Delta_{\mathrm{def}}(A_{i-1}, X)$.

The direct translation *$\Delta(\varphi)$ of the* **Xor***-derivation φ is defined as the string of CNF formulae $[\mathcal{D}(A_0)] : \alpha_1 : \ldots : \alpha_n$.*

Corollary 6.16 (Correctness of direct translations of EXor-derivations):
Let $\varphi = [A_0, \ldots, A_n]$ be an **EXor***-derivation of A_n from A_0 sharp in V. Consider strings of CNF formulae α_i for every $1 \leq i \leq n$ defined as in Definition 6.15, and quantities s_i such that $\ell(\alpha_i) \leq s_i$. Then, the direct translation $\Delta(\varphi)$ of φ is a* **Rat***-derivation of the direct encoding $\mathcal{D}(A_n)$ of the last affine formula A_n in φ from the direct encoding $\mathcal{D}(A_0)$ of the first affine formula A_0 in φ. Furthermore, $\Delta(\varphi)$ is sharp in V and its length is bounded by $\sum_{i=1}^{n} s_i$.*

Proof. We show by induction the following claim:

> For every $0 \leq i \leq n$, the string of CNF formulae $[\mathcal{D}(A_0)] : \alpha_1 : \ldots : \alpha_i$ is a **Rat**-derivation of $\mathcal{D}(A_i)$ from $\mathcal{D}(A_0)$ sharp in V of length bounded by $\sum_{j=1}^{i} s_j$.

The case $i = 0$ is straightforward: the string $[\mathcal{D}(A_0)]$ is an **At**-derivation of $\mathcal{D}(A_0)$ from itself of length $0 = \sum_{j=1}^{0} s_j$. In particular, it is a **Rat**-derivation sharp in $\emptyset$. We now show the induction step. Let us assume that the claim holds for some $0 \leq i < n$; we show it for $i + 1$. There are three possibilities:

a) If $A_{i+1} \subseteq A_i$, then $\alpha_{i+1} = \Delta_{\mathrm{del}}(A_i, A_{i+1})$ and Theorem 6.14 shows that α_{i+1} is an **At**-derivation of $\mathcal{D}(A_{i+1})$ from $\mathcal{D}(A_i)$ (in particular a **Rat**-derivation sharp in $\emptyset$, therefore in V).

b) If $A_{i+1} = A_i \cup \{Y \oplus Z\}$ for some parity constraints $Y, Z \in A_i$, then $\alpha_{i+1} = \Delta_{\mathrm{add}}(A_i, X, Y)$, and by Theorem 6.7 α_{i+1} is an **At**-derivation of $\mathcal{D}(A_i \cup \{X\}) = \mathcal{D}(A_{i+1})$ from $\mathcal{D}(A_i)$. As above, it is a **Rat**-derivation sharp in V as well.

c) If $A_{i+1} = A_i \cup \{X\}$ for some XOR definition X in A_i, then $\alpha_{i+1} = \Delta_{\mathrm{def}}(A_i, X)$. Furthermore, since φ is sharp in V, there must be a variable $a \in V$ occurring in X but not in A_i. Then, Theorem 6.12 shows that α_{i+1} is a **Rat**-derivation of $\mathcal{D}(A_i \cup \{X\}) = \mathcal{D}(A_{i+1})$ from $\mathcal{D}(A_{i+1})$ sharp in $\{a\}$. Since $\{a\} \subseteq V$, it is sharp in V as well.

In all three cases we conclude the same: α_{i+1} is a **Rat**-derivation of $\mathcal{D}(A_{i+1})$ from $\mathcal{D}(A_i)$ sharp in V. By the induction hypothesis and Proposition 3.34, $[\mathcal{D}(A_0)] : \alpha_1 : \ldots : \alpha_{i+1}$ is a **Rat**-derivation of $\mathcal{D}(A_{i+1})$ from $\mathcal{D}(A_0)$ sharp in V. Furthermore, if the length of $[\mathcal{D}(A_0)] : \alpha_1 : \ldots : \alpha_i$ is bounded by $\sum_{j=1}^{i} s_j$ and the length of α_{i+1} is bounded by s_{j+1}, then their concatenation has length bounded by $\sum_{j=1}^{i+1} s_j$.

By induction, the claim holds for all $0 \leq i \leq n$. In particular, for $i = n$, the Corollary follows. □

6.3 Contributions

Direct translations of **EXor**-derivations through the direct encoding have been presented in this chapter. The construction can be obtained, as defined in Definition 6.15, by translating every inference in the original derivation independently and then concatenating the obtained translations. For every inference in the **EXor** proof system, a translation is given in Definitions 6.8, 6.11 and 6.13.

We have as well shown bounds for the length of direct translations. These bounds are tight, in the sense that unless a clause introduced in a CNF formula was already present, the equality in the bounds holds. This means that the length of direct translations can be computed prior to generation of the translation. The

relevance of these bounds lies in the fact that we provide two methods for translation, the other being linear translations explained in Chapter 7, and in practice one may overperform the other in terms of derivation length. Accurate bounds allows to compute translation length for both methods and then use the shortest translation.

This is especially important since, as per Proposition 6.9 and Theorems 6.12 and 6.14, lengths of translated **Xor** inferences are exponentially-sized. Two sources can be identified for the blow up. Firstly, since every inference $A \Rightarrow_{\mathbf{Xor}} B$ is translated through the direct encoding, the result is a **Rat**-derivation of $\mathcal{D}(B)$ from $\mathcal{D}(A)$. The exponential size of direct encodings (see Proposition 5.4) implies that an exponential number of clauses must be introduced into or deleted from $\mathcal{D}(A)$ to obtain $\mathcal{D}(B)$. Secondly, translation of parity constraint addition inferences requires the derivation of an exponential number of intermediate clauses. In order to avoid this exponential behavior, a different method for generating translations of **Xor**-derivations with better complexity properties is presented next.

Chapter 7

Linear translation of XOR derivations

The exponential blow up in direct translations motivates the development of a further method for translating **Xor**-derivations through the direct encoding. *Linear translations* avoid the exponential increase in length experienced by direct translations by bounding the size of involved parity constraints in the original derivation. To do so, the original **Xor**-derivation is first translated through the splitting into an **EXor**'derivation of the splitting of conclusions from the splitting of premises. Parity constraints along this *intermediate translation* have size at most 5.

The intermediate translation is then translated again through the direct encoding using the direct translation. Since parity constraints in the intermediate translation have uniformly bounded size, the exponential blow up in the direct translation is eliminated. This process results in a translation through the linear encoding, which we call a *lift*. That is, we obtain a **Rat**-derivation of the linear encoding of conclusions from the linear encoding of premises. To turn this into a translation through the direct encoding, additional derivations from and into direct encodings are appended to the lift.

As discussed in Section 8.1.1, the asymptotical length of linear translations is exponentially shorter than that of direct translations. Yet, direct translations perform noticeably better than linear translations for input **Xor**-derivations where the ω-measure remains small.

This chapter is structured as follows. The prefix derivation, which derives the linear encoding of an affine formula from its direct encoding, is constructed in Section 7.1. The intermediate derivation and its direct translation, the lift, are presented in Section 7.2. Finally, the suffix derivation, is introduced in Section 7.3, together with the full linear translation. Our results are discussed in Section 7.4.

An outline of linear translations. Similarly to direct translations, *linear translations* map every **EXor**-derivation φ into a translation $\Lambda(\varphi)$ through the direct encoding. That is, $\Lambda(\varphi)$ has as premises the direct encodings of premises for φ, and similarly for conclusions. However, the strategy to generate linear translations is different from that used for direct translations. The main idea consists on first generating an **EXor** translation $\Sigma(\varphi)$ through the splitting, which we call *intermediate translation*. The intermediate translation derives the splitting of conclusions for φ from the splitting of premises for φ. As for the direct translation, the intermediate translation is constructed by translating every inference in the original derivation φ, and then concatenating all partial translations together, as shown in Figure 7.1a.

The intermediate translation is an **EXor**-derivation, and consequently it can be applied the direct translation Δ to obtain a **Rat**-derivation $\Lambda_{\text{lift}}(\varphi)$, which we refer to as the *lift* of φ. The advantage of generating an intermediate derivation is that parity constraints occurring in it have bounded size. This is due to splittings containing parity constraints of size at most 3. Bounded-sized parity constraints in the intermediate derivation become utterly important when applying Δ, since the length of direct translation of individual inferences is exponential on the ω-measure. Bounding parity constraint sizes in $\Sigma(\varphi)$ avoids the exponen-

tial blow up in the direct translation. Unfortunately, the lift itself not qualify as a translation through the direct encoding, but rather as a translation through the linear encoding: premises for $\Lambda_{\text{lift}}(\varphi)$ are the linear encodings of premises for φ, and similarly for conclusions, as can be seen from Figure 7.1a.

This is circumvented by appending two additional derivations, called prefix and suffix derivations.

- The *prefix* derivation $\Lambda_{\text{pre}}(A)$ derives the linear encoding of an affine formula A from its direct encoding. The prefix derivation is obtained in a similar way to the lift. First an **EXor**-derivation, called the *splitter*, is generated. The splitter has premises of φ as premises, and their splitting as conclusions. As for any other **EXor**-derivation, it can be applied the direct translation mapping Δ to obtain a **Rat**-derivation deriving the linear encoding of A from its direct encoding.
- The *suffix* derivation $\Lambda_{\text{suf}}(B)$ does the opposite: it derives the direct encoding of an affine formula B from its linear encoding. Although suffix derivations can be obtained following a process similar to the prefix derivation, we provide a shorter, straightforward construction.

The derivation resulting from prepending the prefix $\Lambda_{\text{pre}}(A_0)$ at the start of the lift $\Lambda_{\text{lift}}(\varphi)$ and appending the suffix $\Lambda_{\text{suf}}(A_n)$ at its end is then a translation of φ through the encoding as we wanted. Figure 7.1b shows an overview of the full construction.

Linear translations are only effective for **Xor**-derivations (i.e. XOR introduction inferences are not translated), but this is all we need to translate derivations produced by Gauss-Jordan elimination (see Theorem 5.12), which is our primary goal. Furthermore, we assume the original **Xor**-derivation to be basic, i.e. containing no Tseitin variables. Otherwise, the splitting of parity constraints along the derivation might not exist. In practice, this is unproblematic: the CNF formula in the solver is assumed to be basic, and **Xor**-derivations, such as those generated by Gauss-Jordan elimination, introduce no additional variables.

Matrix derivations. Before the three parts of linear translations are introduced, we present a result that will be used in their construction. The matrix $\mathcal{M}(X)$ of a parity constraint X, introduced in Section 5.1, merely contains its structure; the constraint is rather imposed by the independent parity constraint of X. Since parity constraints in $\mathcal{M}(X)$ only fix the semantics of splitting variables, it is possible to introduce them as XOR definitions. This is done by the *matrix derivation*.

Definition 7.1 (Matrix derivation):
Let X be a basic parity constraint. Let us enumerate $\text{vars}(X)$ *as $\{a_1, \dots, a_n\}$ with $a_1 < \cdots < a_n$. The matrix of X is the affine formula containing the following parity constraints:*

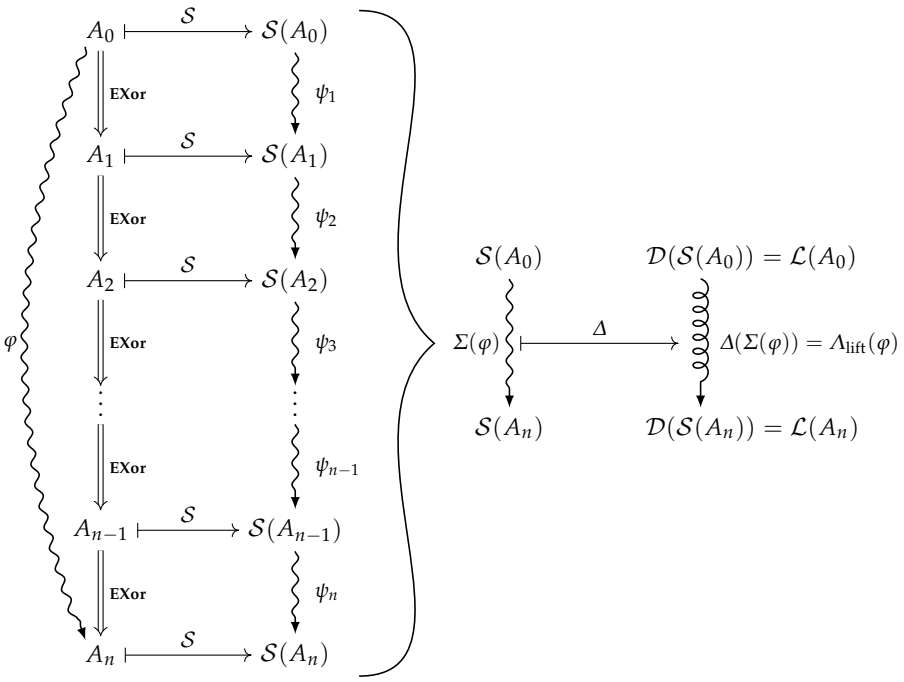
$$Y_0 = \mathsf{par}(\mathsf{u}_0^X) \qquad Y_1 = \mathsf{par}(\mathsf{u}_0^X, a_1, \mathsf{u}_1^X) \qquad \dots \qquad Y_n = \mathsf{par}(\mathsf{u}_{n-1}^X, a_n, \mathsf{u}_n^X)$$

Consider affine formulae $B_i = \{Y_0, \dots, Y_i\}$ for each $0 \leq i \leq n$. We define its matrix derivation *as a string of affine formulae $\Gamma(X) = [\emptyset, B_0, \dots, B_n]$.*

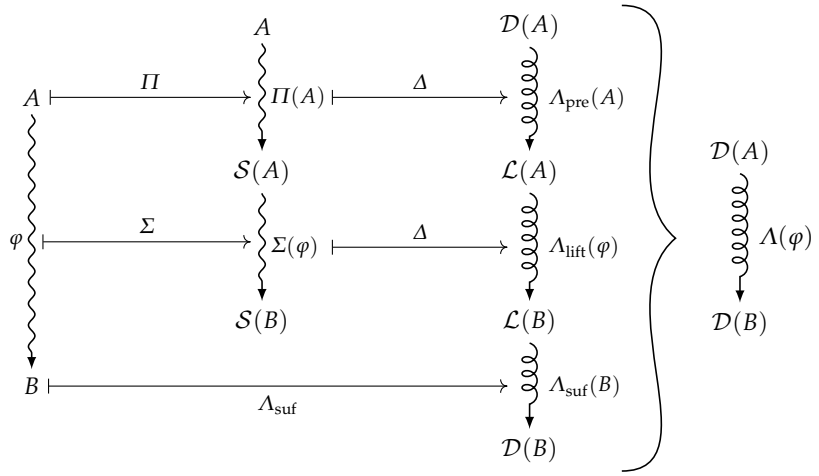
Proposition 7.2 (Correctness of matrix derivations):
Let X be a basic parity constraint. Then, its matrix derivation $\Gamma(X)$ is an **EXor***-derivation of $\mathcal{M}(X)$ from $\emptyset$ sharp in* $\mathbf{SVar}(X)$*. Furthermore, the length of its direct translation $\Delta(\Gamma(X))$ is bounded by $1 + 4|X|$.*

Proof. To show the first claim, we prove the following:

- $\emptyset \Rightarrow_{\mathbf{EXor}} B_0$ *is a XOR definition introduction inference upon* u_0^X this is straightforward, since B_0 only contains the parity constraint $\mathsf{par}(\mathsf{u}_0^X)$, and trivially $\mathsf{u}_0^X \notin \text{vars}(\emptyset)$.
- $B_{i-1} \Rightarrow_{\mathbf{EXor}} B_i$ *is a XOR definition introduction inference upon* u_i^X *for every* $1 \leq i \leq n$ observe that $B_i = B_{i-1} \cup \{Y_i\}$. The variable u_i^X occurs in the parity constraint Y_i, but it does not occur in any of the $Y_0, \dots, Y_{i-1}$, which are the parity constraints included in B_i. Therefore, Y_i is an XOR definition in B_{i-1}.



(a) Construction of the intermediate translation $\Sigma(\varphi)$ and the lift $\Lambda_{\text{lift}}(\varphi)$ of an **EXor** derivation φ. Each inference $A_{i-1} \Rightarrow_{\textbf{EXor}} A_i$ is translated into a **Rat**-derivation ψ_i of $\mathcal{S}(A_i)$ from $\mathcal{S}(A_{i-1})$. Concatenation of the ψ_i yields the intermediate translation $\Sigma(\varphi)$, which is then transformed into the lift $\Lambda_{\text{lift}}(\varphi)$ using the direct translation mapping Δ.



(b) Construction of the full linear translation $\Lambda(\varphi)$ of an **Xor**-derivation φ of B from A. First, the splitter $\Pi(A)$ of the premises A, and the intermediate translation $\Sigma(\varphi)$ of the **Xor**-derivation φ are generated as **EXor**-derivations. Then, the prefix $\Lambda_{\text{pre}}(A)$ and the lift $\Lambda_{\text{lift}}(\varphi)$ are obtained as **Rat**-derivations by direct translation of the splitter and the intermediate translation respectively. The suffix $\Lambda_{\text{suf}}(B)$ is generated as a **Rat**-derivation directly from the conclusion B. Concatenation of the prefix, the lift and the suffix yields the linear translation $\Lambda(\varphi)$ of the **Xor**-derivation φ.

Figure 7.1: Generation of the lift $\Lambda_{\text{lift}}(\varphi)$ and the linear translation $\Lambda(\varphi)$ from an **Xor**-derivation φ. As usual, $\Rightarrow$ arrows represent inferences, "wave" arrows represent **EXor**-derivations, "coil" arrows represent **Rat**-derivations and $\mapsto$ arrows represent the application of a mapping. Furthermore, braces denote concatenation of derivations.

- $B_n = \mathcal{M}(X)$ *holds* straightforward from Definition 7.1.

Then, $\Gamma(X)$ is an **EXor**-derivation of $\mathcal{M}(X)$ from $\emptyset$ sharp in $\{\mathsf{u}_0^X, \ldots, \mathsf{u}_n^X\} \subseteq \mathbf{SVar}(X)$.

To show the bound for the length of the direct translation of $\Gamma(X)$, observe that the direct translation of the first inference step $\emptyset \Rightarrow_{\mathbf{EXor}} B_0$ is $[\emptyset, \mathcal{D}(B_0)]$, and $\mathcal{D}(B_0)$ only contains the parity constraint $\mathsf{par}(\mathsf{u}_0^X)$, whose direct encoding has cardinality $2^{1-1} = 1$. Furthermore, the inference step $B_i \Rightarrow B_{i-1}$ for each $1 \leq i \leq n$ is directly translated into $[\mathcal{D}(B_{i-1}), \mathcal{D}(B_i)]$, and this involves introducing as an unresolvable set of resolution asymmetric tautologies the CNF formula $\mathcal{D}(Y_i)$, whose cardinality is $|\mathcal{D}(Y_i)| = 2^{3-1} = 4$. Since there are $n = |X|$ of the latter inferences, the bound follows. □

7.1 Prefix derivation of an affine formula

In this section, we present a construction for the splitter and prefix derivations. Given an affine formula A, we provide an **EXor**-derivation of the splitting $\mathcal{S}(A)$ from A, which is the *splitter*. The prefix derivation, which derives $\mathcal{L}(A)$ from $\mathcal{D}(A)$ is then obtained by direct translation from the splitter. In Figure 7.1b, the construction of the splitter $\Pi(A)$ for A is denoted by the mapping Π. The *prefix* $\Lambda_{\mathrm{pre}}(A)$ is then constructed applying the splitter $\Pi(A)$ the direct translation mapping Δ.

An outline of the splitter. We first give a sketch on how the splitter is constructed. Given an affine formula A, our goal is to derive the splitting $\mathcal{S}(A)$ from A. A splitter can be constructed for each parity constraint in $X \in A$, that is, an **EXor**-derivation of its splitting $\mathcal{S}(X)$ from X itself. Along the splitter of each parity constraint X, XOR definitions are only introduced upon splitting variables for X. Sharpness sets for generated splitters are then disjoint from $\mathrm{vars}(A)$, and so splitters for individual parity constraints can be fused together into a splitter for A. Here we introduce an approximation to the construction of splitters for individual parity constraints.

Let us assume for this sketch that X has polarity 1 and its variables are enumerated by $\{a_1, \ldots, a_n\}$ with $a_1 < \cdots < a_n$. Then, the splitting of $X = \mathsf{par}(a_1, \ldots, a_n, \top)$ is:

$$\mathcal{S}(X) = \{\mathsf{par}(\mathsf{u}_0^X)\ ,\ \mathsf{par}(\mathsf{u}_0^X, a_1, \mathsf{u}_1^X)\ ,\ \ldots\ ,\ \mathsf{par}(\mathsf{u}_{n-1}^X, a_n, \mathsf{u}_n^X)\ ,\ \mathsf{par}(\mathsf{u}_n^X, \top)\} = \mathcal{M}(X) \cup \{\mathsf{par}(\mathsf{u}_n^X, \top)\}$$

The matrix of X, denoted by $\mathcal{M}(X)$ contains all parity constraints above except the independent parity constraint $\mathsf{par}(\mathsf{u}_n^X, \top)$. The splitter for a parity constraint X is generated by first constructing the matrix derivation $\Gamma(X)$ from Definition 7.1, which derives the matrix $\mathcal{M}(X)$ from $\emptyset$ by introducing all parity constraints in the matrix as XOR definitions in a certain order. If we now are able to derive the independent parity constraint $\mathsf{par}(\mathsf{u}_n^X, \top)$ from $\{X\} \cup \mathcal{M}(X)$, then we will have an **EXor**-derivation of the splitting of X from $\{X\}$ as desired.

The independent parity constraint $\mathsf{par}(\mathsf{u}_n^X, \top)$ can be derived as the addition of X and all parity constraints in the matrix of X.

$$\mathsf{par}(\mathsf{u}_n^X, \top) = \mathsf{par}(a_1, \ldots, a_n, \top) \oplus \mathsf{par}(\mathsf{u}_0^X) \oplus \mathsf{par}(\mathsf{u}_0^X, a_1, \mathsf{u}_1^X) \oplus \cdots \oplus \mathsf{par}(\mathsf{u}_{n-1}^X, a_n, \mathsf{u}_n^X)$$

To justify this, observe that every pseudovariable in the splitting other than u_n^X and $\top$ occurs exactly twice in the right hand side of the equation. Therefore, the independent parity constraint $\mathsf{par}(\mathsf{u}_n^X, \top)$ can be derived by iteratively adding to X every parity constraint in the matrix, which in turn can be derived by XOR definition.

Example 7.3 (Splitter of a parity constraint):

Let us consider the parity constraint $X = \mathsf{par}(\mathsf{p}_1, \mathsf{p}_2, \top)$. The splitting of X is:

$$\mathcal{S}(X) = \{\mathsf{par}(\mathsf{u}_0^X)\ ,\ \mathsf{par}(\mathsf{u}_0^X, \mathsf{p}_1, \mathsf{u}_1^X)\ ,\ \mathsf{par}(\mathsf{u}_1^X, \mathsf{p}_2, \mathsf{u}_2^X)\ ,\ \mathsf{par}(\mathsf{u}_2^X, \top)\}$$

The process described above is reflected in the following addition tree.

$$\dfrac{\dfrac{\dfrac{\mathsf{par}(\mathsf{p}_1, \mathsf{p}_2, \top) \qquad \dfrac{}{\mathsf{par}(\mathsf{u}_0^X)}\,\mathsf{def}}{\mathsf{par}(\mathsf{u}_0^X, \mathsf{p}_1, \mathsf{p}_2, \top)}\,\oplus \qquad \dfrac{}{\mathsf{par}(\mathsf{u}_0^X, \mathsf{p}_1, \mathsf{u}_1^X)}\,\mathsf{def}}{\mathsf{par}(\mathsf{u}_1^X, \mathsf{p}_2, \top)}\,\oplus \qquad \dfrac{}{\mathsf{par}(\mathsf{u}_1^X, \mathsf{p}_2, \mathsf{u}_2^X)}\,\mathsf{def}}{\mathsf{par}(\mathsf{u}_2^X, \top)}\,\oplus$$

Observe that the only premise in this proof sketch is X, and all parity constraints in the splitting $\mathcal{S}(X)$ are derived at some point: parity constraints in the matrix $\mathcal{M}(X)$ are derived by XOR definition introduction, whereas the independent parity constraint $\mathsf{par}(\mathsf{u}_2^X, \top)$ is derived at the bottom of the tree. ■

Expressing this process as an **EXor**-derivation yields a derivation of the splitting $\mathcal{S}(X)$ from the singleton affine formula $\{X\}$. Direct translation yields a **Rat**-derivation of the linear encoding $\mathcal{D}(\mathcal{S}(X)) = \mathcal{L}(X)$ from the direct encoding $\mathcal{D}(X)$.

Constructing the splitter. We now present the formal construction of the splitter. Let A be a basic affine formula, i.e. an affine formula containing no Tseitin variables and therefore no splitting variables, and enumerate parity constraints in A as $A = \{X_1, \ldots, X_n\}$. We provide an **EXor**-derivation of the splitting $\mathcal{S}(A)$ from the parity constraint A. This is done in three steps, depicted in Figure 7.2:

1. For every parity constraint X_i, construct the **EXor** matrix derivation $\Gamma(X_i)$ of the matrix of X_i from the empty formula.
2. For every parity constraint X_i, construct the splitter of X_i, i.e. an **EXor**-derivation ψ_i of the splitting of X_i from $\{X_i\}$.
3. Construct the splitter of A, i.e. an **EXor**-derivation of the splitting of A from A, by fusing together all the ψ_i derivations.

The construction of the matrix derivation in Step 1 was already explained in Proposition 7.2. Observe that, if we can guarantee that the fusion $\{X_i\} \sqcup \Gamma(X_i)$ is an **EXor**-derivation, then it derives $\{X_i\} \cup \mathcal{M}(X_i)$ from $\{X_i\}$. Step 2 is attained by appending additional inference steps to this derivation to derive the only parity constraint from $\mathcal{S}(X_i)$ missing in $\mathcal{M}(X_i)$, i.e. the independent parity constraint of X_i. If variables in X_i are given by $\mathsf{vars}(X_i) = \{a_1, \ldots, a_r\}$ with $a_1 < \cdots < a_r$, then the independent parity constraint is $\mathsf{par}\big(\mathsf{u}_r^{X_i}\big)_{(X_i)}$. As presented in the sketch, the independent parity constraint can be attained by adding all parity constraints in $\{X_i\} \cup \mathcal{M}(X_i)$ together. Let us formalize this by enumerating the parity constraints in the matrix of X_i as

$$Y_0 = \mathsf{par}\big(\mathsf{u}_0^{X_i}\big) \qquad\qquad Y_j = \mathsf{par}\big(\mathsf{u}_{j-1}^{X_i}, a_j, \mathsf{u}_j^{X_i}\big) \quad \text{for } 1 \leq j \leq r$$

and considering the affine formulae

$$B_j = \{X_i\} \cup \mathcal{M}(X_i) \cup \left\{ X_i \oplus \bigoplus_{t=0}^{j} Y_t : 0 \leq t \leq j \right\}$$

for every $0 \leq j \leq r$. Then, define the strings of affine formulae:

$$\chi_i = [\{X_i\} \cup \mathcal{M}(X_i)\ ,\ B_0\ ,\ \ldots\ ,\ B_r\ ,\ \mathcal{S}(X_i)] \qquad\qquad \psi_i = (\{X_i\} \sqcup \Gamma(X_i)) : \chi_i$$

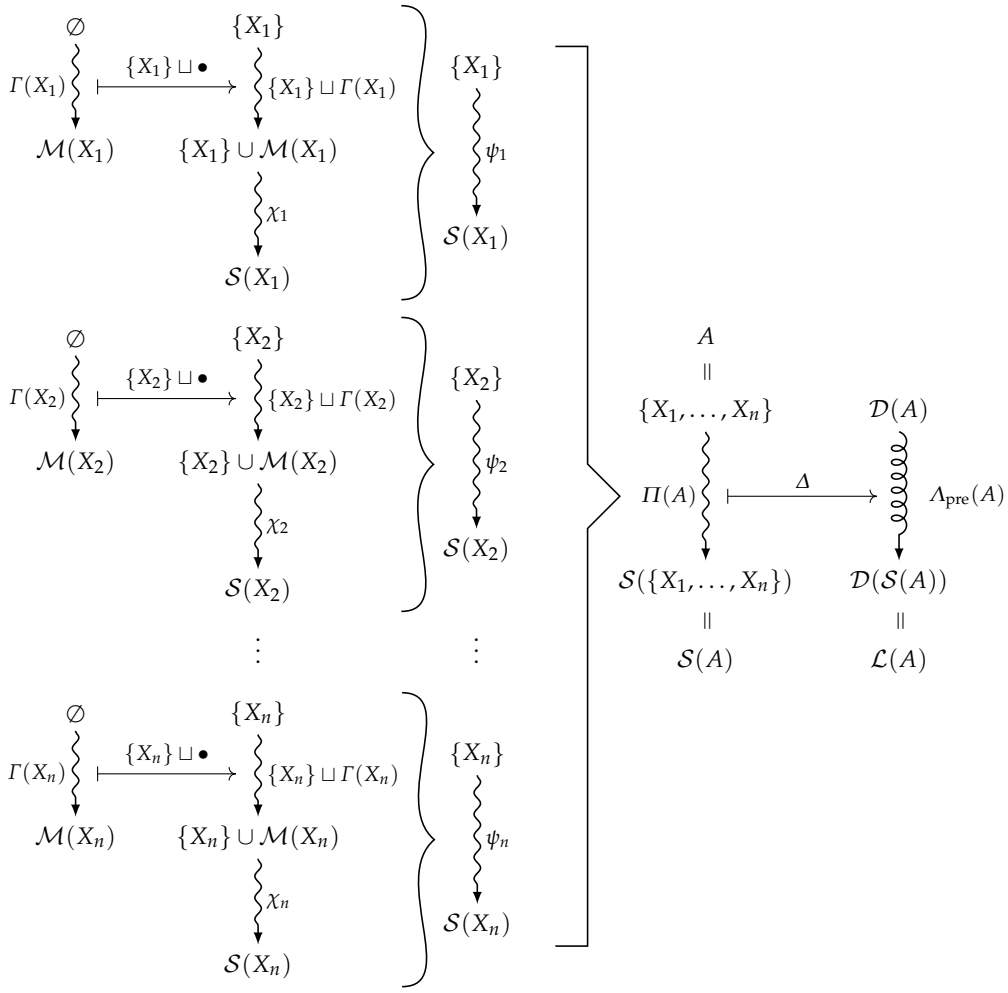


Figure 7.2: Generation of the splitter $\Pi(A)$ and the prefix $\Lambda_{\text{pre}}(A)$ of an affine formula $A = \{X_1, \ldots, X_n\}$. For every parity constraint X_i, the matrix derivation $\Gamma(X_i)$ derives the matrix of X_i from $\emptyset$. Fusing each matrix derivation with the singleton affine formula $\{X_i\}$ yields **EXor**-derivations of $\mathcal{M}(X_i) \cup \{X_i\}$ from $\{X_i\}$. These fusions can be completed by concatenation with derivations χ_i that derive the missing independent parity constraint, thus producing the **EXor**-derivations ψ_i of the splitting of every X_i from $\{X_i\}$. Fusing all the ψ_i together, the splitter $\Pi(A)$ of A is obtained. The prefix $\Lambda_{\text{pre}}(A)$ is then the direct translation of $\Pi(A)$. As usual, "wave" arrows represent **EXor**-derivations, "coil" arrows represent **Rat**-derivations and $\mapsto$ arrows represent the application of a mapping. Furthermore, "curly" braces denote concatenation of derivations, where as "square" brackets represent fusion of derivations.

We now show that ψ_i is an **EXor**-derivation with the properties required for Step 2.

Lemma 7.4 (Correctness of splitters for individual parity constraints):
The following hold for all $1 \leq i \leq n$*:*

i) $\{X_i\} \sqcup \Gamma(X_i)$ *is an* **EXor***-derivation of* $\{X_i\} \cup \mathcal{M}(X_i)$ *from* $\{X_i\}$ *sharp in* $\mathbf{SVar}(X_i)$*. Furthermore, the length of the direct translation* $\Delta(\{X_i\} \sqcup \Gamma(X_i))$ *is bounded by* $1 + 4|X_i|$*.*

ii) χ_i *is an* **Xor***-derivation of* $\mathcal{S}(X_i)$ *from* $\{X_i\} \cup \mathcal{M}(X_i)$*. Furthermore, the length of the direct translation* $\Delta(\chi_i)$ *is bounded by* $8 \cdot 2^{|X_i|} - 7$*.*

iii) ψ_i *is an* **EXor***-derivation of* $\mathcal{S}(X_i)$ *from* $\{X_i\}$ *sharp in* $\mathbf{SVar}(X_i)$*. Furthermore, the length of the direct translation* $\Delta(\psi_i)$ *is bounded by* $8 \cdot 2^{|X_i|} + 4|X_i| - 6$*.*

Proof.

i) By Proposition 7.2, $\Gamma(X_i)$ is an **EXor**-derivation of $\mathcal{M}(X_i)$ from $\emptyset$ sharp in $\mathbf{SVar}(X_i)$. The claim for the fusion follows by Proposition 3.35 by observing that X_i is a basic parity constraint and therefore $\mathrm{vars}(X_i)$ is disjoint from $\mathbf{SVar}(X_i)$. The length bound on the direct translation follows from the corresponding bound for the matrix derivation in Proposition 7.2, together with Proposition 3.41.

ii) To show this claim, we need to check that every inference along χ_i holds.

- $\{X_i\} \cup \mathcal{M}(X_i) \Rightarrow_{\mathbf{Xor}} B_0$ *holds* observe that B_0 contains the parity constraints in $\{X_i\} \cup \mathcal{M}(X_i)$ together with $X_i \oplus Y_0$. Now, $Y_0 = \mathsf{par}\big(\mathsf{u}_0^{X_i}\big)$ occurs in the matrix of X_i, so the inference is covered in **Xor** by parity constraint addition.
- $B_{j-1} \Rightarrow_{\mathbf{Xor}} B_j$ *holds for every* $1 \leq j \leq r$ follows from observing that B_j contains exactly the parity constraints in B_{j-1} together with:

$$X_i \oplus \bigoplus_{t=0}^{j} Y_t = \left(X_i \oplus \bigoplus_{t=0}^{j-1} Y_t \right) \oplus \mathsf{par}\big(\mathsf{u}_{j-1}^{X_i}, a_j, \mathsf{u}_j^{X_i}\big)$$

  The decomposition above yields the claim by parity constraint addition, since the parity constraints $X_i \oplus \bigoplus_{t=0}^{j-1} Y_t$ and $\mathsf{par}\big(\mathsf{u}_{j-1}^{X_i}, a_j, \mathsf{u}_j^{X_i}\big)$ are both included in B_{j-1}.
- $B_r \Rightarrow_{\mathbf{Xor}} \mathcal{S}(X_i)$ *holds* the splitting $\mathcal{S}(X_i)$ consists of the parity constraints in $\mathcal{M}(X_i)$, which are all in B_r, plus the independent parity constraint $\mathsf{par}\big(\mathsf{u}_r^{X_i}\big)_{(X_i)}$. If we show that the latter is in B_r as well, the inference follows by parity constraint deletion. To attain this, we prove the following property by induction on t:

  > For all $0 \leq t \leq r$, the addition $Z_t = X_i \oplus \bigoplus_{t=0}^{j} Y_t$ is the parity constraint $\mathsf{par}\big(\mathsf{u}_j^{X_i}, a_{j+1}, \ldots, a_r\big)_{(X_i)}$, that is, variables in Z_t are exactly $\big\{\mathsf{u}_j^{X_i}, a_{j+1}, \ldots, a_r\big\}$ and the polarity of Z_t is that of X_i.

  In the base case $t = 0$, we have $Z_0 = X_i \oplus Y_0 = X_i \oplus \mathsf{par}\big(\mathsf{u}_0^{X_i}\big)$, and the property follows straightforward.

  Now assume the property holds for some $0 \leq t \leq r$; we show that it holds for $t + 1$ as well. Recall that $Y_{t+1} = \mathsf{par}\big(\mathsf{u}_t^{X_i}, a_{t+1}, \mathsf{u}_{t+1}^{X_i}\big)$. Using the induction hypothesis, we have that variables in Z_{t+1} are given by:

$$\mathrm{vars}(Z_{t+1}) = \mathrm{vars}(Z_t) \triangle \mathrm{vars}(Y_{t+1}) = \big\{\mathsf{u}_t^{X_i}, a_{t+1}, \ldots, a_r\big\} \triangle \big\{\mathsf{u}_t^{X_i}, a_{t+1}, \mathsf{u}_{t+1}^{X_i}\big\} = \\ = \big\{\mathsf{u}_{t+1}^{X_i}, a_{t+2}, \ldots, a_r\big\}$$

and furthermore $\mathrm{pol}(Z_{t+1}) \approx \mathrm{pol}(Z_t) + \mathrm{pol}(Y_{t+1}) = \mathrm{pol}(X_i) + 0 = \mathrm{pol}(X_i)$. The property above follows then for $t+1$.

By induction, the property holds for all $0 \leq t \leq r$, and in particular we have for $t = r$ that $Z_r = \mathsf{par}\big(\mathsf{u}_r^{X_i}\big)_{(X_i)}$. Now observe from the definition of Z_r that this parity constraint occurs in B_r. Thus, the inference follows.

We now show the length bound for the direct translation. Observe that, along the derivation, the following introductions and deletions of parity constraints are performed:

- $Z_0 = \mathsf{par}\big(\mathsf{u}_0^{X_i}, a_1, \ldots, a_r\big)_{(X_i)}$ is introduced as the addition of X_i and $Y_0 = \mathsf{par}\big(\mathsf{u}_0^{X_i}\big)$. Since variables in X_i are disjoint from variables in Y_0, then the length of the direct translation of this inference is given by $2^{|X_i \oplus Y_0|-1} = 2^{r+1-1} = 2^r$.
- For every $1 \leq j \leq r$, the parity constraint $Z_j = \mathsf{par}\big(\mathsf{u}_j^{X_i}, a_{j+1}, \ldots, a_r\big)_{(X_i)}$ is introduced as the addition of $Z_{j-1} = \mathsf{par}\big(\mathsf{u}_{j-1}^{X_i}, a_j, \ldots, a_r\big)_{(X_i)}$ and $Y_j = \mathsf{par}\big(\mathsf{u}_{j-1}^{X_i}, a_j, \mathsf{u}_j^{X_i}\big)$. In this case, we have $\omega(Z_{j-1}, Y_j) = r - j - 3$ and $\delta(Z_{j-1}, Y_j) = r - j + 1$, which yields a length in the direct translation of this inference of $2^{r-j+3} - \frac{3}{2} \cdot 2^{r-j+1}$.
- For every $1 \leq j \leq r$, the parity constraint $Z_{j-1} = \mathsf{par}\big(\mathsf{u}_{j-1}^{X_i}, a_j, \ldots, a_r\big)_{(X_i)}$ is deleted. The direct encoding of this constraint has length 2^{r-j+1}, so this is the length of the direct translation of this inference.

Therefore, the length of $\Delta(\chi_i)$ is bounded by:

$$2^r + \sum_{j=1}^{r} 2^{r-j+3} - \frac{3}{2} \sum_{j=1}^{r} 2^{r-j+1} + \sum_{j=1}^{r} 2^{r-j+1} \tag{7.1}$$

We express the quantities above as geometric sums and apply Proposition 2.3:

$$\sum_{j=1}^{r} 2^{r-j+3} = \sum_{j=3}^{r+2} 2^j = 2^{r+3} - 2^3 = 2^{r+3} - 8$$

$$\frac{3}{2} \sum_{j=1}^{r} 2^{r-j+1} = \frac{2^1 + 2^0}{2^1} \sum_{j=1}^{r} 2^j = \frac{2^1 + 2^0}{2^1}(2^{r+1} - 2) = (2^1 + 2^0)(2^r - 2^0) = 2^{r+1} + 2^r - 3$$

$$\sum_{j=1}^{r} 2^{r-j+1} = \sum_{j=1}^{r} 2^j = 2^{r+1} - 2$$

Then, the expression (7.1) is:

$$2^r + 2^{r+3} - 8 - 2^{r+1} - 2^r + 3 + 2^{r+1} - 2 = (2^0 + 2^3 - 2^1 - 2^0 + 2^1) \cdot 2^r + (3 - 2 - 8) = 8 \cdot 2^r - 7$$

iii) Follows straightforward from the two previous claims by considering that ψ_i being an **Xor**-derivation implies that an ψ_i is an **EXor**-derivation sharp in $\emptyset$, thus in $\mathbf{SVar}(X_i)$. The length bound follows from the addition of the former two. □

Lemma 7.4 shows that the strings ψ_i are **EXor**-derivations of $\mathcal{S}(X_i)$ from $\{X_i\}$ in accordance to Figure 7.2. As we show in Theorem 7.6, their fusion is again an **EXor**-derivation, which gives the splitter needed to acomplish Step 3.

Definition 7.5 (Splitter of an affine formula):
Let $A = \{X_1, \ldots, X_n\}$ be a basic affine formula. Consider **EXor***-derivations ψ_i of the splittings $\mathcal{S}(X_i)$ from $\{X_i\}$ as described through this section for every $1 \leq i \leq n$. We define the* splitter *of A, denoted by $\Pi(A)$, as the string of affine formulae $\psi_1 \sqcup \cdots \sqcup \psi_n$.*

Theorem 7.6 (Correctness of splitters of affine formulae):
Let A be a basic affine formula. Then, $\Pi(A)$ is an **EXor***-derivation of $\mathcal{S}(A)$ from A sharp in* $\mathbf{SVar}(A)$*. Furthermore, the length of its direct translation is bounded by* $\sum_{X\in A}(8\cdot 2^{|X|}+4|X|-6)$

Proof. Let us enumerate $A=\{X_1,\ldots,X_n\}$ and consider **Rat**-derivations ψ_i of $\mathcal{S}(X_i)$ from $\{X_i\}$ as in Definition 7.5. By Lemma 7.4 we know that ψ_i is sharp in $\mathbf{SVar}(X_i)$.

We now check that conditions to apply Proposition 3.37 hold. Let $1\leq i,j\leq n$ be distinct. Since $\{X_1,\ldots,X_n\}$ was an enumeration of A, then $X_i\neq X_j$. Observe now that $\mathrm{vars}(X_j)\cup\mathrm{vars}(\mathcal{S}(X_j))$ is contained in $\mathrm{vars}(X_j)\cap\mathbf{SVar}(X_j)$, and the latter is disjoint from $\mathbf{SVar}(X_i)$. Then, Proposition 3.37 is applicable and the string of affine formulae $\psi_1\sqcup\cdots\sqcup\psi_n=\Pi(A)$ is an **EXor**-derivation of $\bigcup_{i=1}^n\mathcal{S}(X_i)=\mathcal{S}(A)$ from $\bigcup_{i=1}^n\{X_i\}=A$ sharp in $\bigcup_{i=1}^n\mathbf{SVar}(X_i)=\mathbf{SVar}(A)$, as we wanted.

The length bound for the direct translation is given by the bound in Lemma 7.4 together with the extension result for fusion of derivations in Proposition 3.41. □

Constructing the prefix. Last, we need to obtain the prefix derivation from the splitter $\Pi(A)$. This can be achieved by simply applying the direct translation presented in Chapter 6 to the splitter, which corresponds to the rightmost part in Figure 7.2.

Definition 7.7 (Prefix of an affine formula):
Let A be a basic affine formula. We define its prefix *derivation as the* **Rat***-derivation $\Lambda_{\mathrm{pre}}(A)$ resulting from the direct translation of its splitter, that is, $\Lambda_{\mathrm{pre}}(A)=\Delta(\Pi(A))$.*

Corollary 7.8 (Correctness of prefixes of affine formulae):
Let A be a basic affine formula. Then, the prefix derivation $\Lambda_{\mathrm{pre}}(A)$ is an **Rat***-derivation of the linear encoding $\mathcal{L}(A)$ of A from its direct encoding $\mathcal{D}(A)$. Furthermore, $\Lambda_{\mathrm{pre}}(A)$ is sharp in* $\mathbf{SVar}(A)$*, and its length is bounded by* $\sum_{X\in A}(8\cdot 2^{|X|}+4|X|-6)$.

Proof. Straightforward from Theorem 7.6 and Corollary 6.16. □

7.2 Lift derivation of a XOR derivation

We tackle now the problem of generating the lift of an **Xor**-derivation φ, represented as $\Lambda_{\mathrm{lift}}(\varphi)$ in Figure 7.1b. As explained in the introduction of this chapter, our strategy involves obtaining an *intermediate* **EXor** *translation* of φ through the splitting; the *lift* derivation is just its direct translation, in a similar way to how the prefix was obtained in Section 7.1 from the splitter. The intermediate translation, denoted by $\Sigma(\varphi)$, is constructed by translating every inference step in φ through the splitting, as presented in Figure 7.1a. Before we formally present the construction, we give an intuitive idea on how it is obtained.

An outline of the intermediate translation. In the introduction of this chapter we explained how an **Xor**-derivation $\varphi=[A_0,\ldots,A_n]$ of A_n from A_0 can be translated into the intermediate **EXor**-derivation $\Sigma(\varphi)$ of $\mathcal{S}(A_n)$ from $\mathcal{S}(A_0)$ by concatenating **EXor**-derivations ψ_i of $\mathcal{S}(A_i)$ from $\mathcal{S}(A_{i-1})$ for every $1\leq i\leq n$ as depicted in Figure 7.1a. This is achieved independently for every case for **Xor** inferences in Definition 3.21. For the parity constraint deletion case this is straightforward, since clause deletion is supported as a **Rat** inference, and whenever $A_i\subseteq A_{i-1}$ holds $\mathcal{S}(A_i)\subseteq\mathcal{S}(A_{i-1})$ holds as well. The parity constraint addition case is worth of a more thorough explanation.

Assume that $A_i=A_{i-1}\cup\{X\}$ for a parity constraint $X=Y\oplus Z$ where $Y,Z\in A_{i-1}$. Let us enumerate $\mathrm{vars}(\{X,Y,Z\})=\mathrm{vars}(Y)\cup\mathrm{vars}(Z)$ as $\{a_1,\ldots,a_h\}$, where the a_j variables have been chosen in such a way that $a_1<a_2<\cdots<a_h$. For the sake of simplicity, in this sketch we assume that $\mathrm{pol}(X)=\mathrm{pol}(Y)=1$ and

$\mathrm{pol}(Z) = 0$, i.e. both X and Y contain $\top$ and Z does not. The parity constraints can be expressed as

$$X = Y \oplus Z = \mathsf{par}(a_{x_1}, \ldots, a_{x_p}, \top) \qquad Y = \mathsf{par}(a_{y_1}, \ldots, a_{y_q}, \top) \qquad Z = \mathsf{par}(a_{z_1}, \ldots, a_{z_r})$$

where the subindices x_j are chosen from $\{1, \ldots, h\}$ with $x_1 < x_2 < \cdots < x_p$, and similarly for the y_j and z_j. Note that there is an unique way to do so. The splittings of X, Y and Z are then given by:

$$\begin{aligned}
\mathcal{S}(X) &= \left\{ \mathsf{par}(\mathsf{u}_0^X) \ , \ \mathsf{par}(\mathsf{u}_0^X, a_{x_1}, \mathsf{u}_1^X) \ , \ \ldots \ , \ \mathsf{par}\big(\mathsf{u}_{p-1}^X, a_{x_p}, \mathsf{u}_p^X\big) \ , \ \mathsf{par}(\mathsf{u}_p^X, \top) \right\} \\
\mathcal{S}(Y) &= \left\{ \mathsf{par}(\mathsf{u}_0^Y) \ , \ \mathsf{par}(\mathsf{u}_0^Y, a_{y_1}, \mathsf{u}_1^Y) \ , \ \ldots \ , \ \mathsf{par}\big(\mathsf{u}_{q-1}^Y, a_{y_q}, \mathsf{u}_q^Y\big) \ , \ \mathsf{par}(\mathsf{u}_q^Y, \top) \right\} \\
\mathcal{S}(Z) &= \left\{ \mathsf{par}(\mathsf{u}_0^Z) \ , \ \mathsf{par}(\mathsf{u}_0^Z, a_{z_1}, \mathsf{u}_1^Z) \ , \ \ldots \ , \ \mathsf{par}(\mathsf{u}_{r-1}^Z, a_{z_r}, \mathsf{u}_r^Z) \ , \ \mathsf{par}(\mathsf{u}_r^Z) \right\}
\end{aligned} \tag{7.2}$$

We aim to generate an **EXor**-derivation of $\mathcal{S}(X)$ from $\mathcal{S}(\{Y, Z\})$. Parity constraints in the matrix of X can be derived by iteratively introducing XOR definitions, as in the matrix derivation $\Gamma(X)$ in Definition 7.1. Such XOR definitions can be introduced for every parity constraint in the splitting of X except for the independent parity constraint $\mathsf{par}(\mathsf{u}_p^X, \top)$, which does not occur in the matrix of X.

We must now derive the parity constraint $\mathsf{par}(\mathsf{u}_p^X, \top)$ by parity constraint addition inferences. Observe that this is the only clause that remains to be derived, and so all other constraints in (7.2) can be used to perform parity constraint addition inferences. In every step from now on, we derive by addition a parity constraint $\mathsf{par}(\mathsf{u}_u^X, \mathsf{u}_v^Y, \mathsf{u}_w^Z)$ we will refer to as *counter*, with the property that in the j-th step (starting from the 0-th step), the three following conditions hold:

- u is the number of variables from $\{a_1, \ldots, a_j\}$ that occur in X.
- v is the number of variables from $\{a_1, \ldots, a_j\}$ that occur in Y.
- w is the number of variables from $\{a_1, \ldots, a_j\}$ that occur in Z.

For example, in the zeroth step, the sets of variables above are empty, and the counter parity constraint is $\mathsf{par}(\mathsf{u}_0^X, \mathsf{u}_0^Y, \mathsf{u}_0^Z)$, because the number of variables from $\emptyset$ in any of the parity constraints X, Y, Z is trivially 0. This parity constraint can be derived as the addition of the leftmost parity constraints in (7.2). In the j-th step, the set of variables in the conditions above is incremented with the variable a_j. Note that, due to $\mathrm{vars}(X) = \mathrm{vars}(Y) \mathbin{\triangle} \mathrm{vars}(Z)$, the variable a_j occurs in exactly two parity constraints among X, Y, Z. Let us assume that it occurs in X and Y, and therefore not in Z. This means that, in the j-th step, quantities u and v must increment by 1, whereas w must stay the same.

If the counter parity constraint derived in the $(j-1)$-th step was $\mathsf{par}(\mathsf{u}_u^X, \mathsf{u}_v^Y, \mathsf{u}_w^Z)$, then we need to derive the parity constraint $\mathsf{par}(\mathsf{u}_{u+1}^X, \mathsf{u}_{v+1}^Y, \mathsf{u}_w^Z)$. Now, since a_j occurs in X and the invariant says that X contains exactly u variables from $\{a_1, \ldots, a_{j-1}\}$, this means that the parity constraint $\mathsf{par}(\mathsf{u}_u^X, a_j, \mathsf{u}_{u+1}^X)$ must necessarily be in the splitting of X. Analogously, we can deduce that the parity constraint $\mathsf{par}(\mathsf{u}_v^Y, a_j, \mathsf{u}_{v+1}^Y)$ occurs in the splitting of Y. Now simply add these two parity constraints to the previous counter parity constraint to obtain the next counter:

$$\mathsf{par}(\mathsf{u}_u^X, \mathsf{u}_v^Y, \mathsf{u}_w^Z) \oplus \mathsf{par}(\mathsf{u}_u^X, a_j, \mathsf{u}_{u+1}^X) \oplus \mathsf{par}(\mathsf{u}_v^Y, a_j, \mathsf{u}_{v+1}^Y) = \mathsf{par}(\mathsf{u}_{u+1}^X, \mathsf{u}_{v+1}^Y, \mathsf{u}_w^Z)$$

This process can be performed similarly for the other cases. In the end, we obtain the counter $\mathsf{par}(\mathsf{u}_p^X, \mathsf{u}_q^Y, \mathsf{u}_r^Z)$. Adding the independent parity constraints for Y and Z yields the independent parity constraint in X:

$$\mathsf{par}(\mathsf{u}_p^X, \mathsf{u}_q^Y, \mathsf{u}_r^Z) \oplus \mathsf{par}(\mathsf{u}_q^Y, \top) \oplus \mathsf{par}(\mathsf{u}_r^Z) = \mathsf{par}(\mathsf{u}_p^X, \top)$$

This is the independent parity constraint that we needed to derive, so we obtain a derivation of $\mathcal{S}(X) = \mathcal{S}(Y \oplus Z)$. Cases with different polarities can be tackled similarly.

Example 7.9 (Intermediate derivation of an addition inference):
Consider two parity constraints $Y = \mathsf{par}(\mathsf{p}_1, \mathsf{p}_2, \mathsf{p}_3, \mathsf{p}_5, \top)$ and $Z = \mathsf{par}(\mathsf{p}_2, \mathsf{p}_3, \mathsf{p}_4)$, and their addition $X = \mathsf{par}(\mathsf{p}_1, \mathsf{p}_4, \mathsf{p}_5, \top)$. Splittings of these constraints are given by:

$$\mathcal{S}(X) = \{\mathsf{par}(\mathsf{u}_0^X)\ ,\ \mathsf{par}(\mathsf{u}_0^X, \mathsf{p}_1, \mathsf{u}_1^X)\ ,\ \mathsf{par}(\mathsf{u}_1^X, \mathsf{p}_4, \mathsf{u}_2^X)\ ,\ \mathsf{par}(\mathsf{u}_2^X, \top)\}$$
$$\mathcal{S}(Y) = \{\mathsf{par}(\mathsf{u}_0^Y)\ ,\ \mathsf{par}(\mathsf{u}_0^Y, \mathsf{p}_1, \mathsf{u}_1^Y)\ ,\ \mathsf{par}(\mathsf{u}_1^Y, \mathsf{p}_2, \mathsf{u}_2^Y)\ ,\ \mathsf{par}(\mathsf{u}_2^Y, \mathsf{p}_3, \mathsf{u}_3^Y)\ ,\ \mathsf{par}(\mathsf{u}_3^Y, \mathsf{p}_5, \mathsf{u}_4^Y)\ ,\ \mathsf{par}(\mathsf{u}_4^Y, \top)\}$$
$$\mathcal{S}(Z) = \{\mathsf{par}(\mathsf{u}_0^Z)\ ,\ \mathsf{par}(\mathsf{u}_0^Z, \mathsf{p}_2, \mathsf{u}_1^Z)\ ,\ \mathsf{par}(\mathsf{u}_1^Z, \mathsf{p}_3, \mathsf{u}_2^Z)\ ,\ \mathsf{par}(\mathsf{u}_2^Z, \mathsf{p}_4, \mathsf{u}_3^Z)\ ,\ \mathsf{par}(\mathsf{u}_3^Z)\}$$

Observe that parity constraints in the matrix of X (i.e. all parity constraints in $\mathcal{S}(X)$ except the independent parity constraint $\mathsf{par}(\mathsf{u}_2^X, \top)$) can be iteratively derived as XOR definitions from left to right. We execute the counter generation process described above to find out what parity constraints must be added to obtain the independent parity constraint.

j	$u(j)$	$v(j)$	$w(j)$	$\mathsf{p}_j \in \mathrm{vars}(X)$	$\mathsf{p}_j \in \mathrm{vars}(Y)$	$\mathsf{p}_j \in \mathrm{vars}(Z)$
0	0	0	0			
		$\mathsf{par}(\mathsf{u}_0^X, \mathsf{u}_0^Y, \mathsf{u}_0^Z) = \mathsf{par}(\mathsf{u}_0^X) \oplus \mathsf{par}(\mathsf{u}_0^Y) \oplus \mathsf{par}(\mathsf{u}_0^Z)$			
1	1	1	0	✓	✓	
		$\mathsf{par}(\mathsf{u}_1^X, \mathsf{u}_1^Y, \mathsf{u}_0^Z) = \mathsf{par}(\mathsf{u}_0^X, \mathsf{u}_0^Y, \mathsf{u}_0^Z) \oplus \mathsf{par}(\mathsf{u}_0^Y, \mathsf{p}_1, \mathsf{u}_1^Y) \oplus \mathsf{par}(\mathsf{u}_0^X, \mathsf{p}_1, \mathsf{u}_1^X)$			
2	1	2	1		✓	✓
		$\mathsf{par}(\mathsf{u}_1^X, \mathsf{u}_2^Y, \mathsf{u}_1^Z) = \mathsf{par}(\mathsf{u}_1^X, \mathsf{u}_1^Y, \mathsf{u}_0^Z) \oplus \mathsf{par}(\mathsf{u}_1^Y, \mathsf{p}_2, \mathsf{u}_2^Y) \oplus \mathsf{par}(\mathsf{u}_0^Z, \mathsf{p}_2, \mathsf{u}_1^Z)$			
3	1	3	2		✓	✓
		$\mathsf{par}(\mathsf{u}_1^X, \mathsf{u}_3^Y, \mathsf{u}_2^Z) = \mathsf{par}(\mathsf{u}_1^X, \mathsf{u}_2^Y, \mathsf{u}_1^Z) \oplus \mathsf{par}(\mathsf{u}_2^Y, \mathsf{p}_3, \mathsf{u}_3^Y) \oplus \mathsf{par}(\mathsf{u}_1^Z, \mathsf{p}_3, \mathsf{u}_2^Z)$			
4	2	3	3	✓		✓
		$\mathsf{par}(\mathsf{u}_2^X, \mathsf{u}_3^Y, \mathsf{u}_3^Z) = \mathsf{par}(\mathsf{u}_1^X, \mathsf{u}_3^Y, \mathsf{u}_2^Z) \oplus \mathsf{par}(\mathsf{u}_1^X, \mathsf{p}_4, \mathsf{u}_2^X) \oplus \mathsf{par}(\mathsf{u}_2^Z, \mathsf{p}_4, \mathsf{u}_3^Z)$			
5	3	4	3	✓	✓	
		$\mathsf{par}(\mathsf{u}_3^X, \mathsf{u}_4^Y, \mathsf{u}_3^Z) = \mathsf{par}(\mathsf{u}_2^X, \mathsf{u}_3^Y, \mathsf{u}_3^Z) \oplus \mathsf{par}(\mathsf{u}_2^X, \mathsf{p}_5, \mathsf{u}_3^X) \oplus \mathsf{par}(\mathsf{u}_3^Y, \mathsf{p}_5, \mathsf{u}_4^Y)$			

Finally, the last counter parity constraint and the independent parity constraints for Y and Z are added:

$$\mathsf{par}(\mathsf{u}_3^X, \mathsf{u}_4^Y, \mathsf{u}_3^Z) \oplus \mathsf{par}(\mathsf{u}_4^Y, \top) \oplus \mathsf{par}(\mathsf{u}_3^Z) = \mathsf{par}(\mathsf{u}_3^X, \top)$$

The result is the independent parity constraint for X, as intended. The computation above can be transformed into an extended XOR proof tree, shown in Figure 7.3. ■

This process can be integrated into the general derivation by fusing the resulting derivation with the splitting of the affine formula A. The full construction of the intermediate translation for addition inferences is illustrated in Figure 7.4.

Translating addition inferences through the splitting. We now solve the problem of formally generating a translation through the splitting for addition inferences. That is, given an affine formula A and two parity constraints $Y, Z \in A$, we give an **EXor**-derivation of $\mathcal{S}(A \cup \{Y \oplus Z\})$ from $\mathcal{S}(A)$. We first tackle a

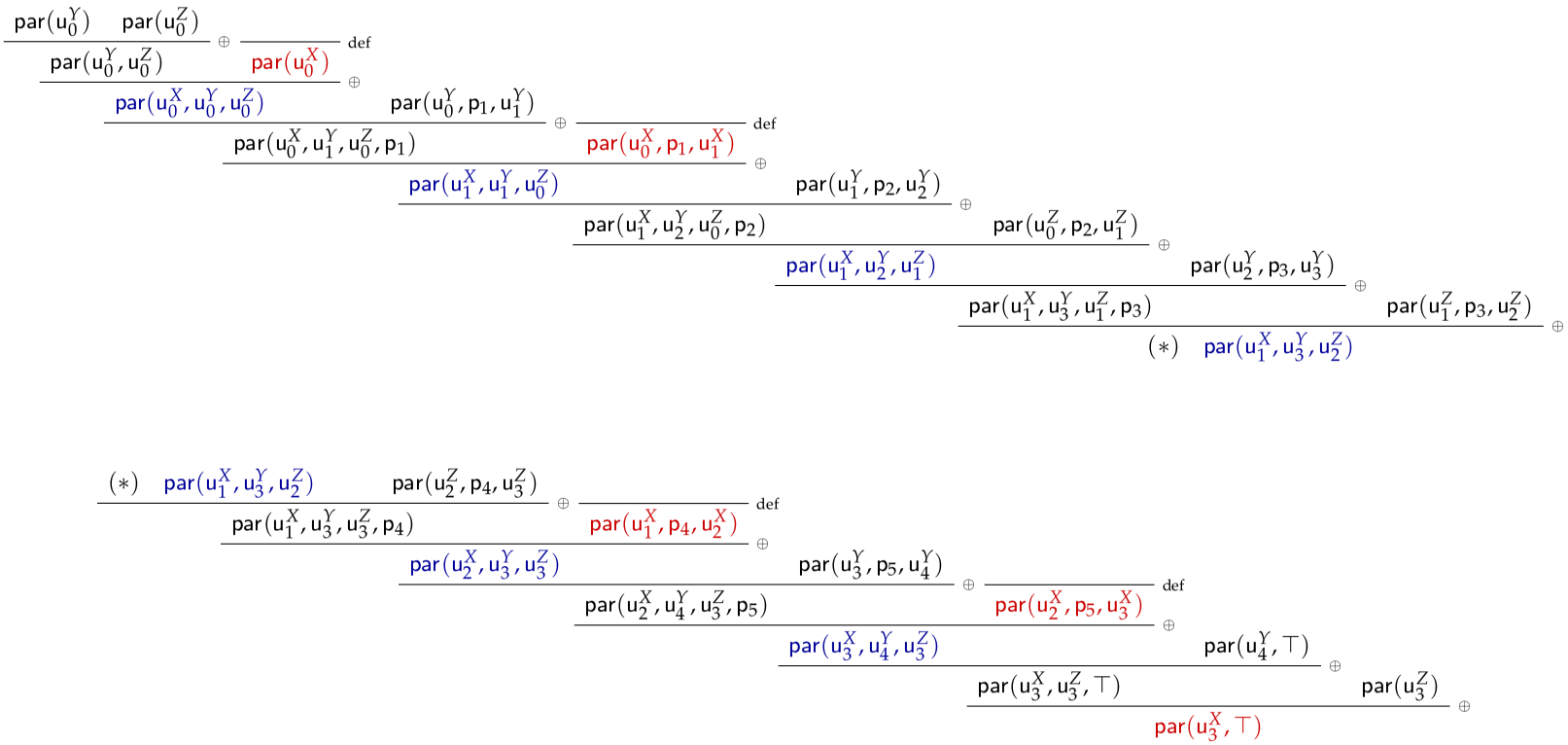


Figure 7.3: Extended XOR tree deriving parity constraints in the splitting $\mathcal{S}(X)$, shown in red, from the splittings $\mathcal{S}(Y)$ and $\mathcal{S}(Z)$. Counter parity constraints are shown in blue. Observe that the tree is shown in two fragments due to space constraints; the merging point is marked by $(*)$.

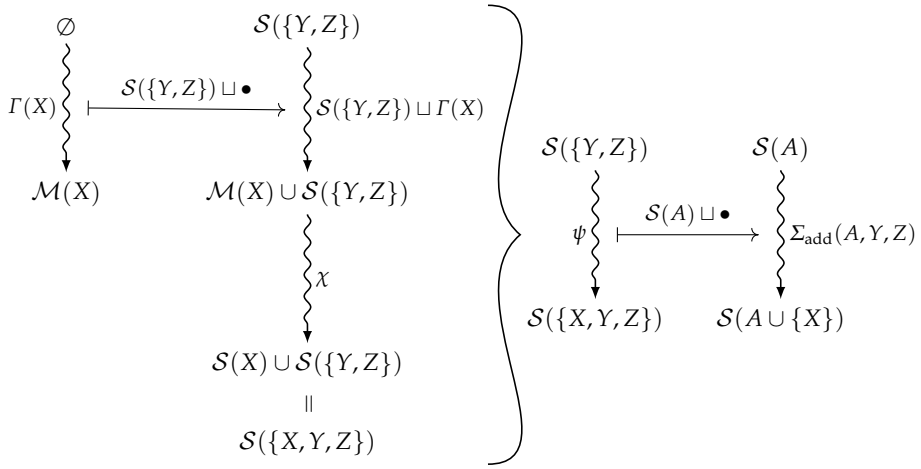


Figure 7.4: Generation of the intermediate translation $\Sigma_{\text{add}}(A, Y, Z)$ of (non-degenerate) parity constraint addition inferences $A \Rightarrow_{\mathbf{EXor}} A \cup \{Y \oplus Z\}$. The matrix of $X = Y \oplus Z$ can be obtained from the matrix derivation $\Gamma(X)$; fusion of the matrix derivation with the splitting of $\{Y, Z\}$ yields an **EXor**-derivation of $\mathcal{M}(X) \cup \mathcal{S}(\{Y, Z\})$. An **EXor**-derivation of the splitting $\mathcal{S}(\{X, Y, Z\})$ from the splitting $\mathcal{S}(\{Y, Z\})$ can be obtained by concatenating this derivation with χ, which derives the missing independent parity constraint of X. Finally, the intermediate translation $\Sigma_{\text{add}}(A, Y, Z)$ is obtained by fusing the latter derivation with the splitting of A. As usual, "wave" arrows represent **EXor**-derivations and $\mapsto$ arrows represent the application of a mapping, where the argument is denoted by $\bullet$. Furthermore, braces denote concatenation of derivations.

degenerate case:

Lemma 7.10 (Correctness of intermediate derivation of addition inferences, degenerate case):
Let A be a basic affine formula and Y and Z be parity constraints in A. If their addition $Y \oplus Z$ occurs in A, then the string of affine formulae $[\mathcal{S}(A)]$ is an **Xor***-derivation of $\mathcal{S}(A) \cup \{Y \oplus Z\}$ from $\mathcal{S}(A)$.*

Proof. Straightforward from $A = A \cup \{Y \oplus Z\}$. □

We now solve the general case where $Y \oplus Z \notin A$. Let $X = Y \oplus Z$ and, as above, enumerate the variables occurring in either X, Y or Z as $\text{vars}(\{X, Y, Z\}) = \{a_1, \ldots, a_h\}$ in such a way that $a_1 < \cdots < a_h$. The splittings are then

$$\begin{aligned}
\mathcal{S}(X) &= \left\{ \mathsf{par}(\mathsf{u}_0^X) \;,\; \mathsf{par}(\mathsf{u}_0^X, a_{x_1}, \mathsf{u}_1^X) \;,\; \ldots \;,\; \mathsf{par}\big(\mathsf{u}_{p-1}^X, a_{x_p}, \mathsf{u}_p^X\big) \;,\; \mathsf{par}(\mathsf{u}_p^X)_{(X)} \right\} \\
\mathcal{S}(Y) &= \left\{ \mathsf{par}(\mathsf{u}_0^Y) \;,\; \mathsf{par}(\mathsf{u}_0^Y, a_{y_1}, \mathsf{u}_1^Y) \;,\; \ldots \;,\; \mathsf{par}\big(\mathsf{u}_{q-1}^Y, a_{y_q}, \mathsf{u}_q^Y\big) \;,\; \mathsf{par}(\mathsf{u}_q^Y)_{(Y)} \right\} \\
\mathcal{S}(Z) &= \left\{ \mathsf{par}(\mathsf{u}_0^Z) \;,\; \mathsf{par}(\mathsf{u}_0^Z, a_{z_1}, \mathsf{u}_1^Z) \;,\; \ldots \;,\; \mathsf{par}\big(\mathsf{u}_{r-1}^Z, a_{z_r}, \mathsf{u}_r^Z\big) \;,\; \mathsf{par}(\mathsf{u}_r^Z)_{(Z)} \right\}
\end{aligned} \tag{7.3}$$

with

$$1 \leq x_1 < \cdots < x_p \leq n \qquad 1 \leq y_1 < \cdots < y_q \leq n \qquad 1 \leq z_1 < \cdots < z_r \leq n$$

Our approach consists of three steps, which are depicted in Figure 7.4.

1. Construct the **EXor** matrix derivation $\Gamma(X)$ of the matrix of X from the empty affine formula.
2. Construct the **EXor**-derivation ψ of $\mathcal{S}(\{X, Y, Z\})$ from $\mathcal{S}(\{Y, Z\})$ by fusing the matrix derivation $\Gamma(X)$ with the splitting of Y and Z, and then appending extra inference steps, given by the derivation χ.

3. Construct the intermediate translation of the parity constraint addition inference, i.e. an **EXor**-derivation of $\mathcal{S}(A \cup \{X\})$ from $\mathcal{S}(A)$ by fusing ψ with $\mathcal{S}(A)$.

The matrix of X can be derived by the matrix derivation $\Gamma(X)$, which is an **EXor**-derivation of $\mathcal{M}(X)$ from $\emptyset$. Furthermore, due to sharpness results, we can fuse this matrix derivation with the affine formula $\mathcal{S}(\{Y, Z\})$, as shown in Figure 7.4. The following result shows that this is sound, and then Step 1 is achieved.

Lemma 7.11 (Correctness of matrix fragments in intermediate derivations):
Let A be a basic affine formula and $Y, Z \in A$ be parity constraints with $X = Y \oplus Z \notin A$. Then, the string of affine formulae $\mathcal{S}(\{Y, Z\}) \sqcup \Gamma(X)$ is an **EXor***-derivation of the affine formula $\mathcal{M}(X) \cup \mathcal{S}(\{Y, Z\})$ from $\mathcal{S}(\{Y, Z\})$ sharp in* $\mathbf{SVar}(X)$*. Furthermore, its direct translation has length bounded by $1 + 4\delta(Y, Z)$.*

Proof. By Proposition 7.2 we know that the matrix derivation $\Gamma(X)$ is an **EXor**-derivation of $\mathcal{M}(X)$ from $\emptyset$ sharp in $\mathbf{SVar}(X)$. Since $X \notin A$, then the splitting $\mathcal{S}(A)$ and $\mathbf{SVar}(X)$ have no variables in common, for A is a basic affine formula. In particular, since $X, Y \in A$, we conclude that $\mathcal{S}(\{X, Y\})$ contains no variables from $\mathbf{SVar}(X)$. By Proposition 3.36, $\mathcal{S}(\{Y, Z\}) \sqcup \Gamma(X)$ is an **EXor**-derivation with the claimed properties. The bound follows from $\delta(Y, Z) = |(|X)$. □

To attain Step 2, we find an **EXor**-derivation χ of $\mathcal{S}(\{X, Y, Z\})$ from the affine formula $B = \mathcal{M}(X) \cup \mathcal{S}(\{Y, Z\})$, and concatenation with the derivation $\mathcal{S}(\{Y, Z\}) \sqcup \Gamma(X)$ from the previous Lemma yields the required derivation ψ. Observe that the only parity constraint from the splitting of $\mathcal{S}(\{X, Y, Z\})$ that is neither in the splittings of Y, Z nor in the matrix of X is the independent parity constraint of X. Our goal is thus to derive $\mathsf{par}(\mathsf{u}_p^X)_{(X)}$ from all other parity constraints in (7.3).

To do so, we define quantities $u(j)$, $v(j)$ and $w(j)$ for every $0 \leq j \leq h$ as follows:

$$u(j) = |\{a_1, \ldots, a_j\} \cap \mathrm{vars}(X)| \qquad v(j) = |\{a_1, \ldots, a_j\} \cap \mathrm{vars}(Y)| \qquad w(j) = |\{a_1, \ldots, a_j\} \cap \mathrm{vars}(Z)|$$

We now iteratively derive affine formulae containing the counter parity constraint W_j defined for every $0 \leq j \leq h$ as $\mathsf{par}\left(\mathsf{u}_{u(j)}^X, \mathsf{u}_{v(j)}^Y, \mathsf{u}_{w(j)}^Z\right)$. We aim to define the following derivations:

- An **Xor**-derivation χ_0 of $B \cup \{W_0\}$ from B.
- An **Xor**-derivation χ_j of $B \cup \{W_j\}$ from $B \cup \{W_{j-1}\}$ for all $1 \leq j \leq h$.
- An **Xor**-derivation χ_{h+1} of $B \cup \{\mathsf{par}(\mathsf{u}_p^X)_{(X)}\} = \mathcal{S}(\{X, Y, Z\})$ from $B \cup \{W_h\}$.

For the first task, the **Xor**-derivation χ_0 is defined as follows:

$$\chi_0 = [B\ ,\ B \cup \{\mathsf{par}(\mathsf{u}_0^X, \mathsf{u}_0^Y)\}\ ,\ B \cup \{\mathsf{par}(\mathsf{u}_0^X, \mathsf{u}_0^Y), W_0\}\ ,\ B \cup \{W_0\}]$$

Let us check that χ_0 is indeed an **Xor**-derivation as desired:

Lemma 7.12 (Correctness of counter fragments, base case):
χ_0 is an **Xor***-derivation of $B \cup \{W_0\}$ from B. Furthermore, the length of the direct translation of χ_0 is 8.*

Proof. We need to prove every inference along the string χ_0.

- $B \Rightarrow_{\mathbf{Xor}} B \cup \{\mathsf{par}(\mathsf{u}_0^X, \mathsf{u}_0^Y)\}$ *holds* we show that the new parity constraint is an addition of two parity constraints in B:
$$\mathsf{par}(\mathsf{u}_0^X, \mathsf{u}_0^Y) = \mathsf{par}(\mathsf{u}_0^X) \oplus \mathsf{par}(\mathsf{u}_0^Y)$$
- $B \cup \{\mathsf{par}(\mathsf{u}_0^X, \mathsf{u}_0^Y)\} \Rightarrow_{\mathbf{Xor}} B \cup \{\mathsf{par}(\mathsf{u}_0^X, \mathsf{u}_0^Y), W_0\}$ *holds* we show that W_0 is an addition of two parity constraints in $B \cup \{\mathsf{par}(\mathsf{u}_0^X, \mathsf{u}_0^Y)\}$:
$$W_0 = \mathsf{par}(\mathsf{u}_0^X, \mathsf{u}_0^Y, \mathsf{u}_0^Z) = \mathsf{par}(\mathsf{u}_0^X, \mathsf{u}_0^Y) \oplus \mathsf{par}(\mathsf{u}_0^Z)$$

- $B \cup \{\mathsf{par}(\mathsf{u}_0^X, \mathsf{u}_0^Y), W_0\} \Rightarrow_{\mathbf{Xor}} B \cup \{W_0\}$ *holds* straightforward, for parity constraint deletion is an inference in **Xor**.

We now check the length bound for the direct translation. Along χ_0 there are the following introductions and deletions of parity constraints:

- $\mathsf{par}(\mathsf{u}_0^X, \mathsf{u}_0^Y)$ is introduced as the addition of $\mathsf{par}(\mathsf{u}_0^X)$ and $\mathsf{par}(\mathsf{u}_0^Y)$. Since their variables are disjoint, the length of the direct translation for this inference is $2^{2-1} = 2$.
- $\mathsf{par}(\mathsf{u}_0^X, \mathsf{u}_0^Y, \mathsf{u}_0^Z)$ is introduced as the addition of $\mathsf{par}(\mathsf{u}_0^X, \mathsf{u}_0^Y)$ and $\mathsf{par}(\mathsf{u}_0^Z)$. Again, their variables are disjoint, so the length of the direct translation for this inference is $2^{3-1} = 4$.
- $\mathsf{par}(\mathsf{u}_0^X, \mathsf{u}_0^Y)$ is deleted. The length of the direct translation for this inference is $2^{2-1} = 2$.

The lengths of partial translations total to 8. □

Now we devise **Xor**-derivations χ_j of $B \cup \{W_j\}$ from $B \cup \{W_{j-1}\}$. It is straightforward to check that, given three sets S_1, S_2, S_3 such that $S_1 \triangle S_2 = S_3$, any element in $S_1 \cup S_2 \cup S_3$ is contained in exactly two of the sets. Therefore, since $\mathrm{vars}(X) = \mathrm{vars}(Y) \triangle \mathrm{vars}(Z)$, the variable a_j occurs in exactly two parity constraints among X, Y and Z. We define two parity constraints W_j' and W_j'' as follows:

i) If $a_j \in \mathrm{vars}(X)$ and $a_j \in \mathrm{vars}(Y)$, which implies $a_j \notin \mathrm{vars}(Z)$, let $W_j' = \mathsf{par}\left(\mathsf{u}^X_{u(j-1)}, a_j, \mathsf{u}^X_{u(j)}\right)$ and $W_j'' = \mathsf{par}\left(\mathsf{u}^Y_{v(j-1)}, a_j, \mathsf{u}^Y_{v(j)}\right)$.

ii) If $a_j \in \mathrm{vars}(Y)$ and $a_j \in \mathrm{vars}(Z)$, which implies $a_j \notin \mathrm{vars}(X)$, let $W_j' = \mathsf{par}\left(\mathsf{u}^Y_{v(j-1)}, a_j, \mathsf{u}^Y_{v(j)}\right)$ and $W_j'' = \mathsf{par}\left(\mathsf{u}^Z_{w(j-1)}, a_j, \mathsf{u}^Z_{w(j)}\right)$.

iii) If $a_j \in \mathrm{vars}(Z)$ and $a_j \in \mathrm{vars}(X)$, which implies $a_j \notin \mathrm{vars}(Y)$, let $W_j' = \mathsf{par}\left(\mathsf{u}^Z_{w(j-1)}, a_j, \mathsf{u}^Z_{w(j)}\right)$ and $W_j'' = \mathsf{par}\left(\mathsf{u}^X_{u(j-1)}, a_j, \mathsf{u}^X_{u(j)}\right)$.

We now show some technical lemmata. Lemma 7.13 essentially shows that parity constraints W_j' and W_j'' occur in the affine formula B. Lemma 7.14 shows that the addition of the counter W_{j-1} with these two parity constraints yields the next counter W_j.

Lemma 7.13 (Structure of splittings):
Let X be an arbitrary parity constraint and $a_1 < \cdots < a_n$ be an increasing sequence of variables containing $\mathrm{vars}(X)$. *Let $1 \leq j \leq n$ be such that $a_j \in \mathrm{vars}(X)$, and let $s = |\mathrm{vars}(X) \cap \{a_1, \ldots, a_j\}|$. Then, the parity constraint* $\mathsf{par}(\mathsf{u}^X_{s-1}, a_j, \mathsf{u}^X_s)$ *occurs in the splitting $\mathcal{S}(X)$.*

Proof. If $a_j \in \mathrm{vars}(X)$, then consider the unique subindices $1 \leq i_1 < \cdots < i_s \leq j$ such that the equality $\mathrm{vars}(X) \cap \{a_1, \ldots, a_j\} = \{a_{i_1}, \ldots, a_{i_s}\}$ holds. Then, for every i_r, the parity constraint $\mathsf{par}(\mathsf{u}^X_{r-1}, a_{i_r}, \mathsf{u}^X_r)$ occurs by Definition 5.5 in $\mathcal{S}(X)$. In particular, $\mathsf{par}(\mathsf{u}^X_{s-1}, a_{i_s}, \mathsf{u}^X_s)$ does. The claim follows from the observation that $i_s = j$, because $a_j \in \mathrm{vars}(X)$ implies that j is the maximum of the subindices $1 \leq r \leq j$ such that $a_r \in \mathrm{vars}(X)$. □

Lemma 7.14 (Evolution of counters):
$W_{j-1} \oplus W_j' \oplus W_j'' = W_j$ *for every* $1 \leq j \leq n$.

Proof. We check cases *(i)*, *(ii)* and *(iii)* in the definition of W_j', W_j'':

i) Since $a_j \notin \mathrm{vars}(Z)$, then $w(j) = w(j-1)$, and we obtain:

$$W_{j-1} \oplus W_j' \oplus W_j'' = \mathsf{par}\big(\mathsf{u}^X_{u(j-1)}, \mathsf{u}^Y_{v(j-1)}, \mathsf{u}^Z_{w(j-1)}\big) \oplus \mathsf{par}\big(\mathsf{u}^X_{u(j-1)}, a_j, \mathsf{u}^X_{u(j)}\big) \oplus \mathsf{par}\big(\mathsf{u}^Y_{v(j-1)}, a_j, \mathsf{u}^Y_{v(j)}\big) =$$
$$= \mathsf{par}\big(\mathsf{u}^X_{u(j)}, \mathsf{u}^Y_{v(j)}, \mathsf{u}^Z_{w(j-1)}\big) = \mathsf{par}\big(\mathsf{u}^X_{u(j)}, \mathsf{u}^Y_{v(j)}, \mathsf{u}^Z_{w(j)}\big) = W_j$$

ii) Since $a_j \notin \mathrm{vars}(X)$, then $u(j) = u(j-1)$, and we obtain:

$$W_{j-1} \oplus W'_j \oplus W''_j = \mathsf{par}\big(\mathsf{u}^X_{u(j-1)}, \mathsf{u}^Y_{v(j-1)}, \mathsf{u}^Z_{w(j-1)}\big) \oplus \mathsf{par}\big(\mathsf{u}^Y_{v(j-1)}, a_j, \mathsf{u}^Y_{v(j)}\big) \oplus \mathsf{par}\big(\mathsf{u}^Z_{w(j-1)}, a_j, \mathsf{u}^Z_{w(j)}\big) = \\ = \mathsf{par}\big(\mathsf{u}^X_{u(j-1)}, \mathsf{u}^Y_{v(j)}, \mathsf{u}^Z_{w(j)}\big) = \mathsf{par}\big(\mathsf{u}^X_{u(j)}, \mathsf{u}^Y_{v(j)}, \mathsf{u}^Z_{w(j)}\big) = W_j$$

iii) Since $a_j \notin \mathrm{vars}(Y)$, then $v(j) = v(j-1)$, and we obtain:

$$W_{j-1} \oplus W'_j \oplus W''_j = \mathsf{par}\big(\mathsf{u}^X_{u(j-1)}, \mathsf{u}^Y_{v(j-1)}, \mathsf{u}^Z_{w(j-1)}\big) \oplus \mathsf{par}\big(\mathsf{u}^Z_{w(j-1)}, a_j, \mathsf{u}^X_{w(j)}\big) \oplus \mathsf{par}\big(\mathsf{u}^X_{u(j-1)}, a_j, \mathsf{u}^X_{u(j)}\big) = \\ = \mathsf{par}\big(\mathsf{u}^X_{u(j)}, \mathsf{u}^Y_{v(j-1)}, \mathsf{u}^Z_{w(j)}\big) = \mathsf{par}\big(\mathsf{u}^X_{u(j)}, \mathsf{u}^Y_{v(j)}, \mathsf{u}^Z_{w(j)}\big) = W_j \qquad \square$$

With these technical results we are now prepared to show that the string

$$\chi_j = \big[B \cup \{W_{j-1}\}\ ,\ B \cup \{W_{j-1}, W_{j-1} \oplus W'_j\}\ ,\ B \cup \{W_{j-1}, W_{j-1} \oplus W'_j, W_j\}\ ,\ B \cup \{W_j\}\big]$$

is an **Xor**-derivation with the desired properties.

Lemma 7.15 (Correctness of counter fragments, induction step):
For every $1 \leq j \leq h$, the string χ_j is an **Xor***-derivation of $B \cup \{W_j\}$ from $B \cup \{W_{j-1}\}$. Furthermore, the length of the direct translation of χ_j is bounded by* 70.

Proof. We need to prove every inference along the string χ_j.

- $B \cup \{W_{j-1}\} \Rightarrow_{\mathbf{Xor}} B \cup \{W_{j-1}, W_{j-1} \oplus W'_j\}$ *holds* Lemma 7.13 implies that $W'_j \in \mathcal{S}(\{X, Y, Z\})$, and since W'_j contains more than one variable, it is not the independent parity constraint $\mathsf{par}(\mathsf{u}^X_p)_{(X)}$ that was excluded from B. Hence, $W'_j \in B$, and the claim follows by parity constraint addition.
- $B \cup \{W_{j-1}, W_{j-1} \oplus W'_j\} \Rightarrow_{\mathbf{Xor}} B \cup \{W_{j-1}, W_{j-1} \oplus W'_j, W_j\}$ *holds* by the same argument as above, $W''_j \in B$. Furthermore, by Lemma 7.14 we know that $W_j = (W_{j-1} \oplus W'_j) \oplus W''_j$, and then the inference holds by parity constraint addition.
- $B \cup \{W_{j-1}, W_{j-1} \oplus W'_j, W_j\} \Rightarrow_{\mathbf{Xor}} B \cup \{W_j\}$ *holds* straightforward, for parity constraint deletion is an inference in **Xor**.

Let us compute the length bound for the direct translation. Along χ_j, the following introductions and deletions of parity constraints are performed:

- $W_{j-1} \oplus W'_j$ is obtain by addition inference of W_{j-1} and W'_j. Observe that $\omega(W_{j-1}, W'_j) = 5$ and $\delta(W_{j-1}, W'_j) = 4$. This yields a length of the direct translation for this inference step of $2^5 - \frac{3}{2} \cdot 2^4 = 8$.
- W_j is obtained as by addition inference of $W_{j-1} \oplus W'_j$ and W''_j. We have $\omega(W_{j-1} \oplus W'_j, W''_j) = 5$ and $\delta(W_{j-1} \oplus W'_j, W''_j) = 3$. This yields a length of the direct translation for this inference step of $2^5 - \frac{3}{2} \cdot 2^3 = 54$.
- $W_{j-1} \oplus W'_j$ is deleted. Since the size of the direct encoding of this parity constraint is $2^{4-1} = 8$, this is the length of the direct translation for this inference step.

The lengths of partial direct translations total to 70. $\square$

At the end of the process described in the previous paragraphs, the counter parity constraint $W_h = \mathsf{par}(\mathsf{u}^X_p, \mathsf{u}^Y_q, \mathsf{u}^Z_r)$ is derived. We now obtain the derivation χ_{h+1} that derives the independent parity constraint $\mathsf{par}(\mathsf{u}^X_p)_{(X)}$ from $B \cup \{W_h\}$. Let us introduce the following notation:

$$X' = \mathsf{par}(\mathsf{u}^X_p)_{(X)} \qquad Y' = \mathsf{par}(\mathsf{u}^Y_q)_{(Y)} \qquad Z' = \mathsf{par}(\mathsf{u}^Z_r)_{(Z)}$$

That is, X' is the independent parity constraint for X, and similarly for Y and Z. Then, χ_{h+1} is a string of affine formulae defined as follows:

$$\chi_{h+1} = [B \cup \{W_h\} \ , \ B \cup \{W_h, W_h \oplus Y'\} \ , \ B \cup \{W_h, W_h \oplus Y', X'\} \ , \ \mathcal{S}(\{X,Y,Z\})]$$

Lemma 7.16 (Correctness of counter fragments, final case):
The string χ_{h+1} is an **Xor***-derivation of $\mathcal{S}(\{X,Y,Z\})$ from $B \cup \{W_h\}$. Furthermore, the length of the direct translation of χ_{h+1} is bounded by 6.*

Proof. We show every inference along the string χ_{h+1}.

- $B \cup \{W_h\} \Rightarrow_{\mathbf{Xor}} B \cup \{W_h, W_h \oplus Y'\}$ *holds* straightforward from observing that the parity constraint Y' is in the splitting of Y, therefore in B. The inference follows by parity constraint addition.
- $B \cup \{W_h, W_h \oplus Y'\} \Rightarrow_{\mathbf{Xor}} B \cup \{W_h, W_h \oplus Y', X'\}$ *holds* note that the parity constraint $Z' = \mathsf{par}(\mathsf{u}_r^Z)_{(Z)}$ occurs in the splitting of Z, so it occurs in B as well. Now, consider the following parity constraint addition:
$$(W_h \oplus Y') \oplus Z' = \mathsf{par}(\mathsf{u}_p^X, \mathsf{u}_q^Y, \mathsf{u}_r^Z) \oplus \mathsf{par}(\mathsf{u}_q^Y)_{(Y)} \oplus \mathsf{par}(\mathsf{u}_r^Z)_{(Z)}$$
The variables in the three parity constraints are
$$\{\mathsf{u}_p^X, \mathsf{u}_q^Y, \mathsf{u}_r^Z\} \qquad \{\mathsf{u}_q^Y\} \qquad \{\mathsf{u}_r^Z\}$$
respectively, which means that their addition contains only one variable, namely u_p^X. Furthermore, the polarities of these parity constraints are 0, $\mathrm{pol}(Y)$ and $\mathrm{pol}(Z)$, respectively; therefore, the polarity of their addition is congruent modulo 2 to $\mathrm{pol}(Y) + \mathrm{pol}(Z) \approx \mathrm{pol}(X)$. Hence, the parity constraint addition above is a parity constraint with set of variables $\{\mathsf{u}_p^X\}$ and the polarity of X, which can only be $X' = \mathsf{par}(\mathsf{u}_p^X)_{(X)}$, and the claim follows.
- $B \cup \{W_h, W_h \oplus Y', X'\} \Rightarrow_{\mathbf{Xor}} \mathcal{S}(\{X,Y,Z\})$ *holds* observe that the parity constraint X' is the only one in the splitting of X missing from B. Therefore, $B \cup \{X'\} = \mathcal{S}(\{X,Y,Z\})$, and then this inference follows by merely parity constraint deletion.

Therefore, χ_{h+1} is an **Xor**-derivation. The rest of the claim for the derivation follows from the observation above that the last affine formula in the derivation is $\mathcal{S}(\{X,Y,Z\})$. Similar considerations as in Lemmata 7.12 and 7.15 yield the length bound. □

We can now obtain the derivation ψ in Step 2, defined as the concatenation of the χ_j. This is shown in Lemma 7.17. Finally, Lemma 7.19 finally yields the intermediate translation $\Sigma_{\mathrm{add}}(A,Y,Z)$ of the addition inference of Y and Z over A, attaining Step 3.

Lemma 7.17 (Correctness of intermediate derivation of addition inferences, regular case):
The string $\psi = (\mathcal{S}(\{Y,Z\}) \sqcup \Gamma(X)) : \chi_0 : \ldots : \chi_{h+1}$ is an **EXor***-derivation of $\mathcal{S}(\{X,Y,Z\})$ from $\mathcal{S}(\{Y,Z\})$ sharp in* $\mathbf{SVar}(X)$.

Proof. Observe in Lemmata 7.12, 7.15 and 7.16 that all the χ_j are **EXor**-derivations sharp in $\mathbf{SVar}(X)$, and furthermore the last affine formula in the derivation χ_{j-1} is the first one in the derivation χ_j. Therefore, the concatenation $\chi = \chi_0 : \ldots : \chi_{h+1}$ is a **Rat**-derivation of $\mathcal{S}(\{X,Y,Z\})$ from $B = \mathcal{M}(X) \cup \mathcal{S}(\{Y,Z\})$. The claim follows from Lemma 7.11. □

Definition 7.18 (Intermediate derivation of addition inferences):
Let A be a basic affine formula and Y, Z be parity constraints in A. The intermediate translation $\Sigma_{\mathrm{add}}(A,Y,Z)$ of the addition inference of Y and Z over A is defined as a string of affine formulae as follows:

i) If $Y \oplus Z \in A$, then $\Sigma_{\mathrm{add}}(A,Y,Z) = [\mathcal{S}(A)]$.

ii) If $Y \oplus Z \notin A$, then $\Sigma_{\text{add}}(A, Y, Z)$ is the fusion $\mathcal{S}(A) \sqcup (\chi_0 : \ldots : \chi_{h+1})$, where the χ_j are defined as previously in this section.

Theorem 7.19 (Correctness of intermediate derivation of addition inferences):
Let A be a basic affine formula and Y, Z be parity constraints in A. Then, the intermediate translation $\Sigma_{\text{add}}(A, Y, Z)$ of the addition inference *of Y and Z over A is an* **EXor***-derivation of $\mathcal{S}(A \cup \{Y \oplus Z\})$ from $\mathcal{S}(A)$ sharp in* $\mathbf{SVar}(Y \oplus Z)$*. Furthermore, the length of the direct translation of $\Sigma_{\text{add}}(A, Y, Z)$ is bounded by 0 if $Y \oplus Z \in A$; and by $15 + 70\omega(Y, Z) + 4\delta(Y, Z)$ otherwise.*

Proof. We distinguish two cases as in Definition 7.18:

- If $Y \oplus Z \in A$, then $\Sigma_{\text{add}}(A, Y, Z) = [\mathcal{S}(A)]$ is by Lemma 7.10 an **Xor**-derivation of $\mathcal{S}(A \cup \{Y \oplus Z\})$ from $\mathcal{S}(A)$, hence an **EXor**-derivation sharp in $\mathbf{SVar}(Y \oplus Z)$. Furthermore, the claim on the length of the direct translation is straightforward, since it is $[\mathcal{L}(A)]$.
- If $Y \oplus Z \notin A$, then $\Sigma_{\text{add}}(A, Y, Z)$ is given by $\mathcal{S}(A) \sqcup \psi$, where ψ is the concatenation in Lemma 7.17, which shows that ψ is an **EXor**-derivation of $\mathcal{S}(\{Y, Z, Y \oplus Z\})$ from $\mathcal{S}(\{Y, Z\})$ sharp in $\mathbf{SVar}(Y \oplus Z)$. Now observe that $\mathbf{SVar}(A)$ only contains splitting variables of the form u_j^W for some $W \in A$ and $j \in \mathbb{N}$. However, $Y \oplus Z \notin A$, so $\text{vars}(A)$ is disjoint from $\mathbf{SVar}(Y \oplus Z)$, and by Proposition 3.36, $\Sigma_{\text{add}}(A, Y, Z)$ is an **EXor**-derivation with the claimed properties.

  To show the bound on the length of the direct translation, just observe that there are exactly $\omega(Y, Z)$ derivations χ_j other than χ_0 and χ_{h+1}. The bound then follows from Lemmata 7.11, 7.12, 7.15 and 7.16 as $(1 + 4\delta(Y, Z)) + 8 + 70\omega(Y, Z) + 6$ □

Constructing the full intermediate translation and the lift. Once single addition inference steps have been assigned an intermediate derivation, constructing the intermediate translation for the whole original derivation is straightforward. We first define intermediate translations for parity constraint deletion inferences as well. Then, we give a formal definition of the full intermediate translation and later we show it to be an **EXor**-derivation as required.

Definition 7.20 (Intermediate derivation of parity constraint deletion inferences):
Let A and B be affine formulae with $B \subseteq A$. We define the intermediate translation $\Sigma_{\text{del}}(A, B)$ of the parity constraint deletion inference *of B over A as the string of affine formulae $[A, B]$.*

Theorem 7.21 (Correctness of intermediate derivation of parity constraint deletion inferences):
Let A and B be affine formulae with $B \subseteq A$. Then, the intermediate translation $\Sigma_{\text{del}}(A, B)$ of the parity constraint deletion inference of B over A is a **Xor***-derivation of $\mathcal{S}(B)$ from $\mathcal{S}(A)$. Furthermore, let s_X be quantities for every $X \in A \setminus B$ defined as:*

$$s_X = 1 \quad \text{, if } X = \mathsf{par}(\emptyset) \qquad s_X = 2 \quad \text{, if } X = \mathsf{par}(\top) \qquad s_X = |X| + 2 \quad \text{otherwise}$$

Then, the length of the direct translation of $\Sigma_{\text{del}}(A, B)$ is bounded by $\sum_{X \in A \setminus B} s_X$.

Proof. The only inference in $\Sigma_{\text{del}}(A, B)$ is straightforward to show by parity constraint deletion. For the length bound, it suffices to observe that s_X is exactly the size of the linear encoding of every parity constraint X deleted in the direct translation of $\Sigma_{\text{del}}(A, B)$. □

Definition 7.22 (Intermediate derivation of Xor-derivations):
Let $\varphi = [A_0, \ldots, A_n]$ be a basic **Xor***-derivation of A_n from A_0. For every $1 \leq i \leq n$, consider strings of affine formulae ψ_i defined as follows:*

i) If $A_i \subseteq A_{i-1}$, then $\psi_i = [\mathcal{S}(A_{i-1}), \mathcal{S}(A_i)]$.

ii) If $A_i = A_{i-1} \cup \{Y \oplus Z\}$ *for some parity constraints* $Y, Z \in A_{i-1}$ *such that* $Y \oplus Z \notin A_{i-1}$ *holds, then* $\psi_i = \Sigma_{\text{add}}(A_{i-1}, Y, Z)$.

Observe that these conditions cover all possible cases for **Xor** *inferences. The* intermediate translation of φ through the splitting, *denoted by* $\Sigma(\varphi)$, *is defined as the concatenation string* $\Sigma(\varphi) = \{\mathcal{S}(A_0)\} : \psi_1 : \ldots : \psi_n$.

Corollary 7.23 (Correctness of intermediate derivation of Xor-derivations):
Let $\varphi = [A_0, \ldots, A_n]$ *be a basic* **Xor***-derivation of the affine formula* A_n *from the affine formula* A_0. *Consider also* **Xor***-derivations* ψ_i *for* $1 \leq i \leq n$ *as defined in Definition 7.22 whose direct translation has length bounded by* s_i. *Then, the intermediate translation* $\Sigma(\varphi)$ *of* φ *is an* **Xor***-derivation of the splitting* $\mathcal{S}(A_n)$ *of* A_n *from the splitting* $\mathcal{S}(A_0)$ *of* A_0. *Furthermore,* $\Sigma(\varphi)$ *is sharp in* **SVar**, *and the length of its direct translation is bounded by* $\sum_{i=1}^{n} s_i$.

Proof. The Theorem follows straightforward from Theorems 7.19 and 7.21. Furthermore, using Proposition 3.41 gives the length bound on the direct translation. □

As shown in Figure 7.1a, now only remains to define the lift of an **EXor**-derivation as the direct encoding of its intermediate translation. Observe that length bounds are directly provided by 7.23.

Definition 7.24:
[Lift of an **Xor***-derivation] Let* φ *be a basic* **Xor***-derivation. The* lift *of* φ *is defined as the string of CNF formulae* $\Lambda_{\text{lift}}(\varphi)$ *given by the direct translation of its intermediate translation,* $\Delta(\Sigma(\varphi))$.

Corollary 7.25 (Correctness of lifts of Xor-derivations):
Let φ *be a basic* **Xor***-derivation of an affine formula* B *from an affine formula* A. *Then, the lift* $\Lambda_{\text{lift}}(\varphi)$ *of* φ *is a* **Rat***-derivation of* $\mathcal{L}(B)$ *from* $\mathcal{L}(A)$ *sharp in* **SVar**.

Proof. Follows straightforward from Corollaries 7.23 and 6.16. □

7.3 Construction of the full linear translation

We now tackle the construction of the full linear translation through the direct encoding. Assume a **Xor**-derivation φ of an affine formula B from another affine formula A is given. In Section 7.2 the lift derivation was obtained, which translates φ through the linear encoding as a **Rat**-derivation of the linear encoding of B from the linear encoding of A.

Since a derivation of the direct encoding of B from the direct encoding of A must be the result of our translation, prefix and suffix derivations that allow to convert into and from the linear encoding are provided. In Section 7.1 a construction for the prefix **Rat**-derivation of the linear encoding of A from the direct encoding of A was presented. To finish our construction, a suffix derivation of the direct encoding of B from the linear encoding of B is constructed.

Constructing the suffix. Constructing the suffix of an affine formula is fortunately easier than doing so for the prefix. Given a basic affine formula B, our goal is to provide an **Rat**-derivation of $\mathcal{D}(B)$ from $\mathcal{L}(B)$. The suffix is given as follows.

Definition 7.26 (Suffix of an affine formula):
Let B *be a basic affine formula. We define its* suffix derivation $\Lambda_{\text{suf}}(B)$ *as the string of CNF formulae*

$$\Lambda_{\text{suf}}(B) = [\mathcal{L}(B)\ ,\ \mathcal{L}(B) \cup \mathcal{D}(B)\ ,\ \mathcal{D}(B)]$$

The reason why the prefix derivation is so simple lies in the fact that clauses in the direct encoding of B are asymmetric tautologies in the linear encoding of B, due to arc-consistency properties of the linear encoding. In our framework, this allows to directly derive all of them in a single inference step.

Lemma 7.27 (Direct encodings are asymmetric tautologies in linear encodings):
Let X be a basic parity constraint and C a clause in its direct encoding $\mathcal{D}(X)$. Then, C is an asymmetric tautology in $\mathcal{L}(X)$.

Proof. Since C is in the direct encoding of X, there is a variable set $V \subseteq \mathrm{vars}(X)$ with $|V| \approx \mathrm{pol}(X) + 1$ such that C can be expressed as the clause $C = \mathsf{nand}(V) \vee \mathsf{or}(\mathrm{vars}(X) \setminus V)$. Let us first enumerate the variables in X as $\mathrm{vars}(X) = \{a_1, \ldots, a_n\}$ in such a way that $a_1 < \cdots < a_n$. The splitting of X is then given by:

$$\mathcal{S}(X) = \{\mathsf{par}(\mathsf{u}_0^X)\ ,\ \mathsf{par}(\mathsf{u}_0^X, a_1, \mathsf{u}_1^X)\ ,\ \ldots\ ,\ \mathsf{par}(\mathsf{u}_{n-1}^X, a_n, \mathsf{u}_n^X), \mathsf{par}(\mathsf{u}_n^X)_{(X)}\}$$

We define the following literals:

$$l_i = \begin{cases} \mathsf{u}_i^X & \text{if } |\{a_1, \ldots, a_i\} \cap V| \approx 0 \\ \neg \mathsf{u}_i^X & \text{if } |\{a_1, \ldots, a_i\} \cap V| \approx 1 \end{cases} \qquad k_i = \begin{cases} a_i & \text{if } a_i \notin V \\ \neg a_i & \text{if } a_i \in V \end{cases}$$

Let $L = \mathrm{lits}(C)$, and observe that $k_i \in L$, since a_i occurs negatively in C if $a_i \in V$, and positively otherwise. We now show by induction on i the following claim:

> For every $0 \leq i \leq n$, the literal l_i is in $\mathrm{ala}^{i+1)}_{\mathcal{L}(X)}(L)$.

We first show the base case $i = 0$. Consider the clause $D = \mathsf{or}(\overline{l_0})$. From the definition of the l_i we can easily check that $l_0 = \mathsf{u}_0^X$. Now, the parity constraint $\mathsf{par}(\mathsf{u}_0^X)$ occurs in the splitting of X, and it is straightforward to check that:

$$D = \mathsf{or}(\neg \mathsf{u}_0^X) = \mathcal{D}(\mathsf{par}(\mathsf{u}_0^X)) \subseteq \mathcal{D}(\mathcal{S}(X)) \subseteq \mathcal{L}(X)$$

Therefore, we have that the literal $\overline{l_0}$ occurs in the clause $D \in \mathcal{L}(X)$, and furthermore any other literal in D occurs in L, which is vacuously true. Therefore, $l_0 \in \mathrm{ala}_{\mathcal{L}(X)}(L)$.

Now we show the induction case. Assume that $l_i \in \mathrm{ala}^{i+1)}_{\mathcal{L}(X)}(L)$ for some $0 \leq i < n$, and consider the clause $D = \mathsf{or}(l_i, k_{i+1}, \overline{l_{i+1}})$. Let us first show that D contains an odd number of negative literals. We distinguish two cases.

- If $a_{i+1} \in V$, then $k_{i+1} = \neg a_{i+1}$ and furthermore we obtain:

  $$|\mathrm{vars}(a_1, \ldots, a_i) \cap V| = |\mathrm{vars}(a_1, \ldots, a_{i+1}) \cap V| + 1$$

  By the definition of the l_j literals, this implies that exactly one of l_i and l_{i+1} is negative. Therefore, the number of negative literals among l_i, k_{i+1} and $\overline{l_{i+1}}$ is either 1 or 3.

- If $a_{i+1} \notin V$, then $k_{i+1} = a_{i+1}$ and

  $$|\mathrm{vars}(a_1, \ldots, a_i) \cap V| = |\mathrm{vars}(a_1, \ldots, a_{i+1}) \cap V|$$

  so l_i and l_{i+1} are either both negative or both positive. Hence, exactly one among l_i, k_{i+1} and $\overline{l_{i+1}}$ is negative.

Now observe that the underlying variables to these three literals are u_i^X, a_{i+1} and u_{i+1}^X. Thus, the clause D is in the direct encoding of the parity constraint $\mathsf{par}(\mathsf{u}_i^X, a_{i+1}, \mathsf{u}_{i+1}^X)$, which is in turn included in the splitting of X. Therefore, $D \in \mathcal{L}(X)$.

Let us write $G = \mathcal{L}(X)$ for typesetting reasons. By induction hypothesis we know that the literal l_i is in $\mathrm{ala}^{i+1)}_G(L)$. Furthermore, we had observed above that $k_{i+1} \in L \subseteq \mathrm{ala}^{i+1)}_G(L)$. This means that a clause D in $\mathcal{L}(X)$ contains the literal $\overline{l_{i+1}}$, and all other literals in D are in $\mathrm{ala}^{i+1)}_G(L)$. Therefore, $l_{i+1} \in \mathrm{ala}^{i+2)}_G(L)$, which is the claim above in the case $i + 1$.

By induction, we conclude that the claim above holds, and in particular $l_n \in \mathrm{ala}_G^{n+1)}(L)$. Consider now the following chain of equivalences:

$$\begin{aligned}
& l_n \text{ is the positive literal } \mathsf{u}_n^X, \\
\text{if and only if } & |\{a_1, \ldots, a_n\} \cap V| \approx 0, \\
\text{if and only if } & |V| \approx 0, \\
\text{if and only if } & \mathrm{pol}(X) \approx 1, \\
\text{if and only if } & \text{the pseudovariable } \top \text{ occurs in the parity constraint } \mathsf{par}(\mathsf{u}_n^X)_{(X)}, \\
\text{if and only if } & \text{the clause } \mathsf{or}(\mathsf{u}_n^X) \text{ occurs in } \mathcal{L}(X)
\end{aligned}$$

Similarly, we can conclude that l_n is the negative literal $\neg\mathsf{u}_n^X$ if and only if the clause $\mathsf{or}(\neg\mathsf{u}_n^X)$ occurs in $\mathcal{L}(X)$. In both cases we conclude that $\mathsf{or}(l_n)$ is in $\mathcal{L}(X)$. In particular, this means that $\overline{l_n}$ occurs in $\mathrm{ala}_G(L)$, hence both literals l_n and $\overline{l_n}$ occur in $\mathrm{ala}_G^{n+1)}(L)$, so this set of literals is inconsistent. Thus, C is an asymmetric tautology in $\mathcal{L}(X)$. □

Corollary 7.28 (Correctness of suffixes of affine formulae):
Let B be a basic affine formula, and consider quantities s_X for every $X \in B$ defined as follows:

$$s_X = 1 \quad \text{, if } X = \mathsf{par}(\emptyset) \qquad s_X = 3 \quad \text{, if } X = \mathsf{par}(\top) \qquad s_X = 2 + 4|X| + 2^{|X|-1} \quad \text{otherwise}$$

Then, the suffix derivation $\Lambda_{\mathrm{suf}}(B)$ is an **At***-derivation of $\mathcal{D}(B)$ from $\mathcal{L}(B)$. Furthermore, the length of $\Lambda_{\mathrm{suf}}(B)$ is bounded by $\sum_{X \in B} s_X$.*

Proof. We need to show that both inferences in the definition of $\Lambda_{\mathrm{suf}}(B)$ hold.

- $\mathcal{L}(B) \Rightarrow_{\mathbf{At}} \mathcal{L}(B) \cup \mathcal{D}(B)$ for every parity constraint $X \in \mathcal{D}(X)$, Lemma 7.27 shows that $\mathcal{D}(X)$ is an asymmetric tautology in $\mathcal{L}(X)$. Therefore, it is an asymmetric tautology in $\mathcal{L}(B)$ as well. We obtain

$$\mathcal{D}(B) = \bigcup_{X \in B} \mathcal{D}(X) \subseteq \mathcal{L}(B)^{\mathsf{AT}}$$

  which shows the inference.

- $\mathcal{L}(B) \cup \mathcal{D}(B) \Rightarrow_{\mathbf{At}} \mathcal{D}(B)$ follows straightforward by parity constraint deletion.

For the length bound, observe that along $\Lambda_{\mathrm{suf}}(B)$ every clause in $\mathcal{D}(B)$ is introduced and every clause in $\mathcal{L}(B)$ is deleted. The bound follows from observing that $s_X = |\mathcal{D}(B)| + |\mathcal{L}(B)|$. □

Combining the prefix, the lift and the suffix. We can, finally, define the linear translation through the direct encoding of an **Xor**-definition and show that it fits our requirements. Observe that length bounds are given by Corollaries 7.8, 7.25 and 7.28.

Definition 7.29 (Linear translation of an Xor-derivation):
Let φ be a basic **Xor***-derivation of an affine formula B from an affine formula A. The* linear translation *of φ is defined as the string of affine formulae $\Lambda(\varphi) = \Lambda_{\mathrm{pre}}(A) : \Lambda_{\mathrm{lift}}(\varphi) : \Lambda_{\mathrm{suf}}(B)$.*

Corollary 7.30 (Correctness of linear translations of Xor-derivations):
Let φ be a basic **Xor***-derivation of an affine formula B from an affine formula A. The linear translation $\Lambda(\varphi)$ of φ is a* **Rat***-derivation of $\mathcal{D}(B)$ from $\mathcal{D}(A)$ sharp in* **SVar***.*

Proof. Throughout this chapter we have obtained the following results:

- $\Lambda_{\mathrm{pre}}(A)$ is a **Rat**-derivation of $\mathcal{L}(A)$ from $\mathcal{D}(A)$ sharp in $\mathbf{SVar}(A)$, by Corollary 7.8. Consequently, it is sharp in **SVar**.
- $\Lambda_{\mathrm{lift}}(\varphi)$ is a **Rat**-derivation of $\mathcal{L}(B)$ from $\mathcal{L}(A)$ sharp in **SVar**, by Corollary 7.25.

- $\Lambda_{\text{suf}}(B)$ is an **At**-derivation of $\mathcal{D}(B)$ from $\mathcal{L}(B)$, by Corollary 7.28. Therefore, $\Lambda_{\text{suf}}(B)$ is a **Rat**-derivation of $\mathcal{D}(B)$ from $\mathcal{L}(B)$ sharp in **SVar**.

The claim follows by Propositions 3.6 and 3.34. □

7.4 Contributions

Linear translations of **Xor**-derivations along the direct encoding have been presented in this chapter. The construction is obtained as the concatenation of three independently generated **Rat**-derivations, as shown in Definition 7.29. First a prefix derivation that derives the linear encoding of the premises is constructed in Definition 7.7 as the direct translation of the splitter, presented in Definition 7.5. The second **Rat**-derivation is the lift, constructed in Definition 7.24, which derives the linear encoding of the conclusions from the linear encoding of the premises. The lift is generated as the direct translation of an intermediate **EXor**-derivation, introduced in Definition 7.22, which translates individual inferences in the original **Xor**-derivation into **EXor**-derivations through the splitting. Finally, the suffix **Rat**-derivation is constructed in Definition 7.26 by straightly deriving the direct encoding of conclusions from their linear encoding.

The advantage of linear translations is that they avoid a large part of the exponential blow up in direct translations by transforming the original **Xor**-derivation into an **EXor**-derivation where the size of involved parity constraints is uniformly bounded. This gives linear bounds to the size of the lift, as shown in Theorems 7.19 and 7.21. However, linear bounds only apply to intermediate parity constraints in the original **Xor**-derivation. As shown in Corollaries 7.8 and 7.28, the prefix and suffix derivations still present an exponential behavior. This is mainly due to the fact that they still need to deal with direct encodings. However, the blow up in linear translations is restricted to premises and conclusions, which implies a large improvement over direct translations, especially when large intermediate parity constraints are generated in the original **Xor**-derivation. We now proceed to both theoretically and empirically compare our two methods for translation.

Chapter 8

Practical issues

In this chapter, we analyze several questions regarding practical application of the techniques developed throughout this Thesis to the generation of unsatisfiability proofs in SAT solvers with integrated parity reasoning. Section 8.1 is devoted to compare the performance of direct and linear translations in terms of translation length. One of the main concerns in unsatisfiability proof generation is how to attain short proofs, because proofs generated by SAT solvers can be remarkably long. To compare lengths of direct and linear translations, we first show an exponential separation between them. In particular, the length of linear translations can be polynomially bounded by the combined size of direct encodings of both premises and conclusions of the original **Xor**-derivation, under the reasonable assumption that the latter has quadratic length on the number of premises. In contrast, some executions of the Gauss-Jordan elimination algorithm yield **Xor**-derivations whose direct translation is only exponentially bounded on the same measure.

However, this argument is only valid as a worst-case separation. We develop an empirical study on the length of both translations of **Xor**-derivations obtained from problems from the SAT Competition 2014. The data suggests a strong relation between the relative performance of direct and linear translations, and the maximum value of the ω-measure along the original **Xor**-derivation. For instances where this measure is big enough, linear translations greatly overperform direct translations. However, in many instances this measure remained small, and then direct translations are significantly better than linear translations, although by a much lower margin than in the opposite case. In practice, this empirical result can be used to determine which approach should be used to generate a translation during solving; and in any case, the length bounds provided along Chapters 6 and 7 allow to approximate the lenght of translations with both methods prior to translation generation, thus enabling the solver to choose the most beneficial approach.

Section 8.2 provides the last results that are needed to apply our translations to practical generation of unsatisfiability proofs. Proofs for the remaining rules in the XOR-CDCL transition system, namely $\xrightarrow[\text{X-learn}]{}$ and $\xrightarrow[\text{X-simp}]{}$ are provided. Finally, the issue of generating DRAT representations using our methods is tackled. Although a technique to obtain them was presented in Section 4.2.2, it requires the generation of the full **Rat**-derivation, which is largely unfeasible. We present results that allow the generation of DRAT representations by composing elementary DRAT representations. Since all **Rat**-derivations in this Thesis are generated by composing smaller **Rat**-derivations, these results allow the efficient generation of unsatisfiability proofs in the DRAT format. Our results are analyzed in Section 8.3.

8.1 Comparing direct and linear translations

In Chapters 6 and 7, direct and linear translations were presented. Both provide methods to translate **Xor**-derivations through the direct encoding. This means that, given an **Xor**-derivation of an affine formula B from an affine formula A, a **Rat**-derivation of the direct encoding of B from the direct encoding of A can

be generated. In general, the main concern in unsatisfiability proof generation is the length of generated derivations. It is then a relevant question which of the two translations perfoms best regarding this issue.

In this section, we provide two comparisons between direct and linear translations. The first one is theoretical: we show that linear translations are linearly-sized on the size (including the size of premises and conclusions) of the original **Xor**-derivation. However, this is far from true for direct translations: there is a family of **Xor**-derivations with exponentially-sized direct translations.

The second comparison is empirical. We show that, despite of our theoretical results, direct translations are still significantly shorter than linear translations in many practical situations. We empirically find conditions for this. In practice, this poses no practical problem, since tight bounds for both direct and linear translations can be determined prior to generation of the derivations, thus allowing the solver to choose for the shorter.

8.1.1 Theoretical lengths of direct and linear translations

We first provide a general bound for the size of linear translations.The obtained bound for the linear translation of a **Xor**-derivation φ of B from A is linear on the sum of the sizes of direct encodings of A and B, and the length of φ. In our practical application, the direct encoding of A is part of the input CNF formula, whereas the direct encoding of B is required to be generated in the output; therefore, these are reasonable measures for the size of the translation. Regarding the length of the **Xor**-derivation φ, it can in principle be arbitrarily long. However, in practice φ is generated by Gauss-Jordan elimination, which means that the length of φ is quadratic in $|A|$, as explained in Section 5.2.

Lemma 8.1 (Auxiliary bounds):
Let X be a parity constraint and A be an affine formula. The following hold:

i) $|X| \leq |\mathcal{D}(X)|$.

ii) $2^{|X|} \leq 2|\mathcal{D}(X)| + 1$.

iii) $|A| \leq |\mathcal{D}(A)| + 1$.

iv) $|\mathrm{vars}(A)| \leq |\mathcal{D}(A)|$.

Proof.

i) The case when $|X| = 0$ is immediate. When $|X| > 0$, we can show the claim by induction on $|X|$. If $|X| = 1$, then $|\mathcal{D}(X)| = 2^{1-1} = 1 \geq |X|$. Assume now that the claim holds for $|X| = n$ for some $n \geq 1$. Then, if $|X| = n+1$ we have $|\mathcal{D}(X)| = 2^{n+1-1} = 2 \cdot 2^{n-1} \geq 2n \geq n+1 = |X|$. The claim then follows by induction.

ii) The cases $X = \mathsf{par}(\emptyset)$ and $X = \mathsf{par}(\top)$ are straightforward to check. In the case when $|X| > 0$, we obtain $2|\mathcal{D}(X)| + 1 = 2 \cdot 2^{|X|-1} + 1 = 2^{|X|} + 1 \geq 2^{|X|}$.

iii) Let $A' = A \setminus \{\mathsf{par}(\emptyset)\}$. Then, we have

$$|\mathcal{D}(A)| + 1 = |\mathcal{D}(A')| + 1 = \left| \bigcup_{Y \in A'} \mathcal{D}(Y) \right| + 1 = \sum_{Y \in A'} |\mathcal{D}(Y)| + 1 \geq |A'| + 1 \geq |A|$$

where we have used that distinct parity constraints have disjoint direct encodings, and $|\mathcal{D}(Y)| \geq 1$ for every parity constraint Y other than $\mathsf{par}(\emptyset)$.

iv) By σ-additivity of the cardinality measure [Hal76] we obtain:

$$|\mathrm{vars}(A)| = \left| \bigcup_{Y \in A} \mathrm{vars}(Y) \right| \geq \sum_{Y \in A} |\mathrm{vars}(Y)| = \sum_{Y \in A} |Y|$$

Now, by Claim *(i)*, we obtain $|\mathrm{vars}(A)| \geq \sum_{Y \in A} |\mathcal{D}(Y)| = |\mathcal{D}(A)|$. □

Theorem 8.2 (Upper bound for linear translations):
Let φ be a basic **Xor***-derivation of an affine formula B from an affine formula A. Then:*

$$\ell(\Lambda(\varphi)) \leq (22|\mathcal{D}(A)| + 2) + (15 + 74|\mathcal{D}(A)|)\ell(\varphi) + 9|\mathcal{D}(B)|$$

In particular, if $\ell(\varphi) \leq k|A|^2$ for some $k \in \mathbb{N}$, then $\ell(\Lambda(\varphi))$ is bounded by $O((|\mathcal{D}(A)| + |\mathcal{D}(B)|)^3)$.

Proof. The **Rat**-derivation $\Lambda(\varphi) = \Lambda_{\mathrm{pre}}(A) : \Lambda_{\mathrm{lift}}(\varphi) : \Lambda_{\mathrm{suf}}(B)$ can be bounded separately for each of the component **Rat**-derivations; the sum of the bounds for $\Lambda_{\mathrm{pre}}(A)$, $\Lambda_{\mathrm{lift}}(\varphi)$ and $\Lambda_{\mathrm{suf}}(B)$ yields the desired bound. Along this proof, we refer to bounds in Lemma 8.1 simply by their claim numbers. We first bound the length of the prefix $\Lambda_{\mathrm{pre}}(A)$. By Corollary 7.8, we obtain:

$$\ell(\Lambda_{\mathrm{pre}}(A)) \leq \sum_{X \in A} \left(8 \cdot 2^{|X|} + 4|X| - 6\right)$$

Applying bounds *(i)* and *(ii)*, we obtain:

$$\ell(\Lambda_{\mathrm{pre}}(A)) \leq \sum_{X \in A} (16|\mathcal{D}(X)| + 8 + 4|\mathcal{D}(X)| - 6) = 2|A| + 20 \sum_{X \in A} |\mathcal{D}(X)|$$

By bound *(iii)* and considering that distinct parity constraints have disjoint direct encodings, we obtain:

$$\ell(\Lambda_{\mathrm{pre}}(A)) \leq 2|\mathcal{D}(A)| + 2 + 20|\mathcal{D}(A)| = 22|\mathcal{D}(A)| + 2$$

We now obtain a bound for the length of the lift $\Lambda_{\mathrm{lift}}(\varphi)$. Assume that $\varphi = [A_0, \ldots, A_n]$, and let us decompose $\Sigma(\varphi)$ as $[A_0] : \psi_1 : \ldots : \psi_n$ as in Definition 7.23. Then, each ψ_i is a **EXor**-derivation of A_i from A_{i-1} for $1 \leq i \leq n$, and we get $\ell(\Lambda_{\mathrm{lift}}(\varphi)) \leq \sum_{i=1}^{n} \ell(\Delta(\psi_i))$. Observe that, for every **Xor** inference $B' \Rightarrow_{\mathbf{Xor}} B''$, it holds that $\mathrm{vars}(B'') \subseteq \mathrm{vars}(B')$. This justifies that $\mathrm{vars}(A_i) \subseteq \mathrm{vars}(A_0) = \mathrm{vars}(A)$ for every $1 \leq i \leq n$. Define $p = |\mathrm{vars}(A)|$. We show the following claim:

For every $1 \leq i \leq n$, we have $\ell(\Delta(\psi_i)) \leq (15 + 74p)\ell(\psi_i)$.

To show this, we distinguish two cases

- If $A_i = A_{i-1} \cup \{Y \oplus Z\}$ for parity constraints $Y, Z \in A_{i-1}$ such that $Y \oplus Z \notin A_i$, then we have $A_i \triangle A_{i-1} = \{Y \oplus Z\}$, so $\ell(\psi_i) = 1$. By Theorem 7.19, we obtain:

  $$\ell(\Delta(\psi_i)) = \ell(\Delta(\Sigma_{\mathrm{add}}(A_{i-1}, Y, Z))) =\leq 15 + 70\omega(Y, Z) + 4\delta(Y, Z)$$

  Note that both $\omega(Y, Z) \leq p$ and $\delta(Y, Z) \leq p$ hold because $Y, Z \in A_{i-1}$ and $\mathrm{vars}(A_{i-1}) \subseteq \mathrm{vars}(A)$. Hence, we obtain $\ell(\Delta(\psi_i)) \leq 15 + 74p \leq (15 + 74p)\ell(\psi_i)$.

- If $A_i \subseteq A_{i-1}$, then $A_i \triangle A_{i-1} = A_{i-1} \setminus A_i$, and hence:

  $$\ell(\Delta(\psi_i)) \leq \sum_{X \in A_{i-1} \setminus A_i} s_X$$

  where s_X is defined as in Definition 7.21 for every $X \in A_{i-1} \setminus A_i$. In particular, $s_X \leq \max\{1, 2, 4|X| + 2\}$, so in any case $s_X \leq 2 + 4|X| \leq 15 + 74p$. Hence, $\ell(\Delta(\psi_i)) \leq (15 + 74p)\ell(\psi_i)$ follows by observing that $\ell(\psi_i) = |A_{i-1} \setminus A_i|$.

The claim above then follows. By bound *(iv)* we obtain:

$$\ell(\Lambda_{\text{lift}}(\varphi)) \leq \sum_{i=1}^{n}(15+74p)\ell(\psi_i) = (15+74p)\sum_{i=1}^{n}\ell(\psi_i) = (15+74|\mathcal{D}(A)|)\ell(\varphi)$$

Last, let us obtain a bound for the length of the suffix $\Lambda_{\text{suf}}(B)$. From Corollary 7.28, we have that $\ell(\Lambda_{\text{suf}}(B)) \leq \sum_{X\in B} s_X$ where $s_X = \max\{1,3,2+4|X|+2^{|X|-1}\}$. In any case, by bounds *(i)* and *(ii)* we obtain:

$$s_X \leq 3+4|X|+\frac{1}{2}2^{|X|} \leq 3+4|\mathcal{D}(X)|+\frac{1}{2}(|\mathcal{D}(B)|+1) \leq 4+5|\mathcal{D}(B)|$$

Hence, we obtain $\ell(\Lambda_{\text{suf}}(B)) \leq \sum_{X\in B}(4+5|\mathcal{D}(B)|) \leq 4|B|+5|\mathcal{D}(B)|$, and by bound *(iii)* we conclude $\ell(\Lambda_{\text{suf}}(B)) \leq 9|\mathcal{D}(B)|$.

Adding the derived bounds we obtain:

$$\ell(\Lambda(\varphi)) = \ell(\Lambda_{\text{pre}}(A)) + \ell(\Lambda_{\text{lift}}(\varphi)) + \ell(\Lambda_{\text{suf}}(B)) \leq (22|\mathcal{D}(A)|+2) + (15+74|\mathcal{D}(A)|)\ell(\varphi) + 9|\mathcal{D}(B)|$$

which is the first claim. To derive the second claim, observe that bound *(iii)* gives $\ell(\varphi) \leq k|\mathcal{D}(A)|^2+k$. Let us denote $|\mathcal{D}(A)|+|\mathcal{D}(B)| = N$. Hence,

$$\ell(\Lambda(\varphi)) \leq (22|\mathcal{D}(A)|+2) + (15+74|\mathcal{D}(A)|)(k|\mathcal{D}(A)|^2+k) + 9|\mathcal{D}(B)| \leq$$
$$\leq (2+15k) + (22+9+74k)N + 15kN^2 + 74N^3 = O(N^3) \qquad \square$$

Observe that the bounds given in Lemma 8.1 are very rough for large values of $|X|$ and $|A|$, and therefore so is the bound provided by Theorem 8.2. Even so, direct translations may become so large that they exceed this bound. The following example shows that, in general, direct translations are only exponentially bound in the combined size of premises and conclusions.

Example 8.3 (Worst-case lower bound for direct translations):
Let a_i and b_i be distinct basic variables for $i \in \mathbb{N}$, and consider parity constraints and affine formulae:

$$\begin{aligned}
X_i &= \mathsf{par}(a_{i-1}, b_i, a_i), \quad \text{for } 1 \leq i \\
Y_{i,j} &= \mathsf{par}(a_i, b_1, \ldots, b_j), \quad \text{for } 0 \leq j \leq i \\
Z_i &= \mathsf{par}(b_i), \quad \text{for } 1 \leq i \\
A_{i,n} &= \{Y_{0,0}, \ldots, Y_{i,i}, X_{i+1}, \ldots, X_n, Z_1, \ldots, Z_n\}, \quad \text{for } 0 \leq i \leq n \\
B_{j,n} &= \{Y_{0,0}, \ldots, Y_{j,j}, Y_{j+1,j}, \ldots, Y_{n,j}, Z_1, \ldots, Z_n\}, \quad \text{for } 0 \leq j \leq n
\end{aligned}$$

Let us fix $n \in \mathbb{N}$. The sequence $A_{0,n}, \ldots, A_{n,n}, B_{n,n}, \ldots, B_{1,n}$ represents steps in a Gauss-Jordan elimination procedure. In particular, $A_{i-1,n}$ is transformed into $A_{i,n}$ by selecting variable a_i in the parity constraint $Y_{i-1,i-1}$ and eliminating it from all other parity constraints in $A_{i-1,n}$. There is only one such parity constraint, namely X_i, so $A_{i,n}$ is obtained by replacing X_i by $Y_{i-1,i-1} \oplus X_i = Y_{i,i}$. The transition from $A_{n,n}$ to $B_{n,n}$ involves no operation, since these affine formulae coincide. Finally, $B_{j,n}$ is transformed into $B_{j-1,n}$ by selecting variable b_j in the parity constraint Z_j and eliminating it from all other parity constraints in $B_{j,n}$ containing it, namely $Y_{j,j}, \ldots, Y_{n,j}$. Observe that $B_{j,n} \oplus Y_{i,j} = Y_{i,j-1}$ for all $0 \leq j \leq i$, so the resultant affine formula from this elimination step is $B_{j-1,n}$. The procedure stops when $B_{0,n}$ is obtained, which is a reduced affine formula.

Let us formalize this as an **Xor**-derivation. The affine formula $A_{i,n}$ is derived from $A_{i-1,n}$ by the **Xor**-derivation $\varphi_i = [A_{i-1,n}\ ,\ A_{i-1,n} \cup \{Y_{i,i}\}\ ,\ A_i]$ for all $1 \leq i \leq n$. Furthermore, the description above can be formalized into **Xor**-derivations ψ_i of $B_{i-1,n}$ from $B_{i,n}$ for all $1 \leq i \leq n$; we do not give explicit expressions for the ψ_i because these are irrelevant for our results. Then, a full **Xor**-derivation of $B_{0,n}$ from $A_{0,n}$ obtained

from a Gauss-Jordan elimination procedure with input $A_{0,n}$ is $\chi_n = \varphi_1 : \ldots : \varphi_n : \psi_n : \ldots : \psi_1$. Observe that along each φ_i exactly one parity constraint is replaced by another, so the length of φ_i is 2. Furthermore, in every ψ_i, a total of $n - i + 1$ parity constraints are replaced, so the length of ψ_i is $2(n - i + 1)$. The total length of χ_n is given by:

$$\ell(\chi_n) = 2\sum_{i=1}^{n}(2 + n - i) = 4n + 2n^2 - n(n+1) = n^2 + 3n$$

Observe that, as required for **Xor**-derivations in Theorem 8.2, $\ell(\chi_n) \leq |A_{0,n}|^2 = (1+2n)^2$. There, an upper bound of $O(N^3)$ where N is the combined size of direct encodings of premises and conclusions is shown for the length of the linear translation. Let us show that χ_n has exponential size on N.

Consider the direct translation of χ_n. A lower bound for its length is given by $\ell(\Delta(\chi_n)) \geq \ell(\Delta(\varphi_n))$. Let us denote $A' = A_{n-1,n-1} \cup \{Y_{n,n}\}$. The **Xor**-derivation φ_n contains two inferences: $A_{n-1,n-1} \Rightarrow_{\mathbf{EXor}} A'$ by parity constraint addition as $Y_{n,n} = Y_{n-1,n-1} \oplus X_n$; and $A' \Rightarrow_{\mathbf{EXor}} A_{n,n}$ by parity constraint deletion of X_n. In particular, the direct translation of φ_n is given by $\Delta(\varphi_n) = \Delta_{\mathsf{add}}(A_{n-1,n-1}, Y_{n-1,n-1}, X_n) : \Delta_{\mathsf{del}}(A', A_{n,n})$. Hence, the length of the direct translation of χ_n has lower bound $\ell(\Delta_{\mathsf{add}}(A_{n-1,n-1}, Y_{n-1,n-1}, X_n))$.

Let us compute the ω- and δ-measures for this parity constraint addition inference:

$$Y_{n-1,n-1} = \mathsf{par}(a_{n-1}, b_1, \ldots, b_{n-1}) \qquad X_n = \mathsf{par}(a_{n-1}, b_n, a_n)$$
$$\omega(Y_{n-1,n-1}, X_n) = n + 2 \qquad \delta(Y_{n-1,n-1}, X_n) = n + 1$$

Then, we obtain:

$$\ell(\Delta(\chi_n)) \geq \ell(\Delta_{\mathsf{add}}(A_{n-1,n-1}, Y_{n-1,n-1}, X_n)) = 2^{n+2} - 2^{n+1} = 2^{n+1}(2^1 - 2^0) = 2^{n+1} \tag{8.1}$$

Using Proposition 5.4, it is straightforward to compute:

$$|\mathcal{D}(A_{0,n})| = |\mathcal{D}(Y_{0,0})| + \sum_{i=1}^{n}|\mathcal{D}(X_i)| + \sum_{i=1}^{n}|\mathcal{D}(Z_i)| = 1 + 4n + n = 5n + 1$$
$$|\mathcal{D}(B_{0,n})| = \sum_{i=1}^{n}|\mathcal{D}(Y_{i,0})| + \sum_{i=1}^{n}|\mathcal{D}(Z_i)| = n + n = 2n$$

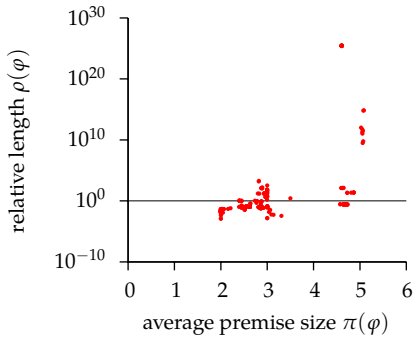
Take $N = |\mathcal{D}(A_{0,n})| + |\mathcal{D}(B_{0,n})| = 7n + 1$. Then, bound (8.1) shows that χ_n is of length exponential on N. ■

Theorem 8.2 and Example 8.3 together provide a strong argument to prefer linear translations over direct translations: the former have polynomial length (and therefore polynomial size if derivations are to be stored in the DRAT format) in the combined size of premises and conclusions, whereas the latter may experience an exponential blow up. The exponential worst-case complexity of direct translations is due to the presence of unbounded intermediate parity constraints in the original **Xor**-derivation. This problem is avoided in linear translations by using bounded-size parity constraints. In fact, the main bottlenecks in linear translations are the prefix and suffix derivations; the lift scales linearly with the ω-measure along the original derivation.

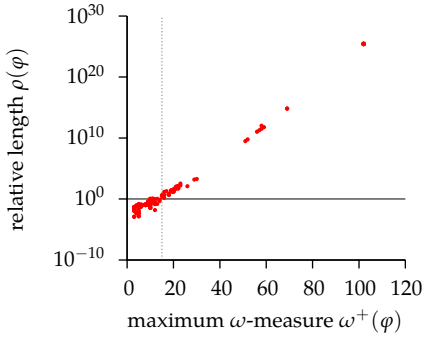
8.1.2 Empirical results on lengths of direct and linear translations

Although theoretically useful, the results in Section 8.1.1 are of an asymptotic and worst-case nature, and it might well be that direct translations perform better than linear translations for practically relevant cases. In this section we present empirical results on the compared sizes of direct and linear translations.

We designed an experiment to test how much shorter are linear translations compared to direct trans-



(a) Length of direct translations relative to length of linear translations compared to the average size of premises for **Xor**-derivations. The horizontal line indicates equal length of linear and direct translations.



(b) Length of direct translations relative to length of linear translations compared to the maximum value of the ω-measure for parity constraint addition inferences for **Xor**-derivations. The horizontal line indicates equal length of linear and direct translations. The vertical line shows the separation in performance at $\omega^{+}(\varphi) = 15$.

Figure 8.1: Results of experiments

lations, and under which conditions do direct translations overperform linear translations. Since direct translations may experience a blow up, instead of generating both translations and then measuring their lenghts, we exploited the tight bounds found along Chapters 6 and 7. As explained there, our bounds are almost exact, in the sense that the actual translation can only be shorter due to repetition of formulae. In practice, since translations are generated by concatenating small subderivations, it is reasonable to assume that the presented bounds and the length of generated derivations coincide. Therefore, we use these bounds to directly compute the lengths of translations without generating them.

Our experimental setup involves obtaining **Xor**-derivations for applications of Gauss-Jordan elimination as inprocessing steps to CNF formulae. Records of Gauss-Jordan elimination steps were obtained from the `CoProcessor` [Man11] preprocessing tool for SAT solving. Due to implementation issues, only premises and addition steps could be obtained; therefore, in our experiments the suffix derivation is disregarded. From these, lengths of translations were computed.

Experiments were run using as benchmark the instances of the application track of the SAT Competition 2014 [BDHJ14]. Out of 300 instances, CoProcessor was able to successfully apply Gauss-Jordan elimination to 210 instances. For each instance, the *relative length* of direct encodings to linear encodings, which can be understood as the proportional overhead increase due to the use of direct translations instead of linear translations, was computed. That is, we computed the following measure:

$$\rho(\varphi) = \frac{|\Delta(\varphi)|}{|\Lambda(\varphi)|}$$

Experiments showed a rather diverse behavior. While many direct translations are about 100 times shorter than their linear counterparts, the latter can be up to 25 orders of magnitude shorter than direct translations. As expected from final remarks in Section 8.1.1, this is not related to parity constraint sizes in the premises, but rather to long parity constraints occurring along the **Xor**-derivation. Figure 8.1a compares the ρ-measure for every benchmark instance to the average size of parity constraints in the premises. If φ is an **Xor**-derivation of B from A, *the average premise size* is:

$$\pi(\varphi) = \frac{\sum_{X \in A} |X|}{|A|}$$

Most instances have an average premise size between 2 and 3, and for those instances the relative length

varies between 10^{-3} and 10^{3} without any noticeable regularity. Besides, when bigger parity constraints occur in the premises, the relative length varies even more wildly: some instances have direct translations 25 orders of magnitude longer than their linear translations, whereas in other instances with large average premise size direct translations slightly overperform linear translations.

In Figure 8.1b, the ρ-measure is compared to the *maximum ω-measure* found along addition inferences in the original **Xor**-derivation for every instance. The latter is formally computed for an **Xor**-derivation $\varphi = [A_0, \ldots, A_n]$ as

$$\omega^+(\varphi) = \max\Big\{\omega(Y,Z) : \text{ for some } 1 \leq i \leq n,\ \ Y \oplus Z \in A_i \setminus A_{i-1}\Big\}$$

i.e. the maximum of the $\omega(Y,Z)$ for parity constraint addition inferences $A_{i-1} \Rightarrow_{\mathbf{Xor}} A_{i-1} \cup \{Y \oplus Z\} = A_i$. We find that the relative length $\rho(\varphi)$ bears a strong exponential relation to $\omega^+(\varphi)$. In particular, our data suggest that linear translation overperforms direct translation whenever the maximum ω-measure is above 15.

The implications of this experiment are rather ambiguous. Both linear and direct translations can be useful, depending on the particular instance. When deciding which approach to use, two criteria may be used. The first one is to compute some measure on the derivation, and heuristically decide for linear or direct translations. For example, as suggested above, one may compute ω^+ for a derivation and choose direct translations if the result is above 15, and linear translations otherwise. The second criterion relies on directly computing the length of both translations prior to their generation, which can be done using the bounds in Chapters 6 and 7, and then decide for the shorter.

8.2 Unsatisfiability proofs for the XOR-CDCL framework

Once translations of **Xor**-derivations through the direct encoding are available, whether they are direct or linear translations, the problem of generating unsatisfiability proofs for XOR-CDCL SAT solving can finally be solved. As in Section 4.2.2, it suffices to give independent derivations of a CNF formula F from another CNF formula F' whenever $\langle F, \nu\rangle \xrightarrow[\text{X-CDCL}]{} \langle F', \nu'\rangle$. Remember that the $\xrightarrow[\text{X-CDCL}]{}$ transition relation was defined as the union of the following transition relations between states:

$$\xrightarrow[\text{unit}]{} \qquad \xrightarrow[\text{decide}]{} \qquad \xrightarrow[\text{back}]{} \qquad \xrightarrow[\text{X-unit}]{} \qquad \xrightarrow[\text{X-learn}]{} \qquad \xrightarrow[\text{X-simp}]{}$$

Derivations for the $\xrightarrow[\text{unit}]{}$, $\xrightarrow[\text{decide}]{}$ and $\xrightarrow[\text{back}]{}$ rules were already provided in Section 4.2.2. Furthermore, the $\xrightarrow[\text{X-unit}]{}$ rule does not modify the CNF formula in the state, so $F = F'$ and the trivial **Rat**-derivation $[F]$ satisfies the requirements. We now tackle the remaining cases. In the rest of this section, we denote by Φ an arbitrary mapping from basic **Xor**-derivations to **Rat**-derivations such that, for all basic **Xor**-derivations of an affine formula B from another affine formula A, $\Phi(\varphi)$ is a **Rat**-derivation of the direct encoding $\mathcal{D}(B)$ from the direct encoding $\mathcal{D}(A)$ sharp in **TVar**.

The X-learn rule Recall that the $\xrightarrow[\text{X-learn}]{}$ rule was defined by $\langle F, \nu\rangle \xrightarrow[\text{X-learn}]{} \langle F \cup \{\mathcal{C}(\sigma)\}, \nu\rangle$ if K is an implication graph XOR-compatible with $\langle F, \nu\rangle$ and $\sigma = [\lambda_0, \ldots, \lambda_n]$ is a shift on K. By using Proposition 4.19, it suffices to give **Rat**-derivations of $\{\mathcal{R}(\lambda_i)\}$ from F for all $0 \leq i \leq n$ to construct a **Rat**-derivation of $F \cup \{\mathcal{C}(\sigma)\}$ from F.

Proposition 8.4 (Unsatisfiability proofs for the $\xrightarrow[\text{X-learn}]{}$ rule):
Let F be a basic CNF formula and ν be a partial assignment. Let K be an implication graph XOR-compatible with $\langle F, \nu\rangle$ and $\sigma = [\lambda_0, \ldots, \lambda_n]$ be a shift on K. For every $0 \leq i \leq n$, let α_i be a string of CNF formulae as follows:

i) If $\mathcal{R}(\lambda_i) \in F$, then $\alpha_i = [F, \{\mathcal{R}(\lambda_i)\}]$.

ii) If there is an affine formula $A = X_1, \ldots, X_p$ with $\mathcal{D}(A) \subseteq F$ such that $\mathcal{X}(\lambda_i) = \bigoplus_{j=1}^{p} X_j$, then consider parity constraints Y_j and affine formulae A_j for $1 \leq j \leq n$ defined as

$$Y_j = \bigoplus_{t=1}^{j} X_t \qquad\qquad A_j = A \cup \{Y_1, \ldots, Y_j\}$$

and define $\varphi_i = [A_1, \ldots, A_p, \{Y_p\}]$ and $\alpha_i = F \sqcup (\Phi(\varphi_i) : [\{\mathcal{R}(\lambda_i)\}])$.

Then, α_i is a **Rat***-derivation of $\{\mathcal{R}(\lambda_i)\}$ from F for all $0 \leq i \leq n$.*

Proof. Case *(i)* holds straightforward by clause deletion. We show case *(ii)*. First we show the following claim:

φ_i is a basic **Xor**-derivation of $\{\mathcal{X}(\lambda_i)\}$ from A.

To show this, we need to prove the following conditions:

- $A = A_1$ *holds* straightforward from $A_1 = A \cup \{Y_1\}$ and $Y_1 = X_1 \in A$.
- $A_{j-1} \Rightarrow_{\mathbf{Xor}} A_j$ *holds for all* $2 \leq j \leq p$ observe that $A_j = A_{j-1} \cup \{Y_j\}$ and $Y_j = Y_{j-1} \oplus X_j$. Hence, the inference follows by parity clause addition introduction.
- $A_p \Rightarrow_{\mathbf{Xor}} \{Y_p\}$ *holds* we know that $Y_p \in A_p$, so the inference follows by parity constraint deletion.
- $\{Y_p\} = \{\mathcal{X}(\lambda_i)\}$ *holds* follows from the condition that $\mathcal{X}(\lambda_i) = \bigoplus_{j=1}^{p} X_j$.

Then, the claim holds, and this implies that $\Phi(\varphi_i)$ is a **Rat**-derivation of $\mathcal{D}(\mathcal{X}(\lambda_i))$ from $\mathcal{D}(A)$ sharp in **TVar**. Furthermore, by Lemma 5.16, if we define $G = \{\mathcal{R}(\lambda_i)\}$ we have that $\Phi(\varphi_i) : [G]$ is a **Rat**-derivation of G from $\mathcal{D}(A)$ sharp in **TVar**. Now, F is a basic CNF formula, so $\mathrm{vars}(F)$ is disjoint from **TVar** and thus the fusion $\alpha_i = F \cup (\Phi(\varphi_i) : [G])$ is a **Rat**-derivation of G from $F \cup \mathcal{D}(A) = F$, as we wanted to show. □

The X-simp rule The $\xrightarrow[\text{X-simp}]{}$ rule is defined by $\langle F \cup \mathcal{D}(A), [] \rangle \xrightarrow[\text{X-simp}]{} \langle F \cup \mathcal{D}(A'), [] \rangle$ if A and A' are affine formulae with $A \equiv A'$. Generally, A' is obtained from A by Gauss-Jordan elimination, so it is safe to assume that an **Xor**-derivation of A' from A is available. In this case, the translation gives directly the desired **Rat**-derivation of $F \cup \mathcal{D}(A')$ from $F \cup \mathcal{D}(A)$ by fusion.

Proposition 8.5 (Unsatisfiability proofs for the $\xrightarrow[\text{X-simp}]{}$ rule):
Let F be a basic CNF formula and A, A' be basic affine formulae with $A \equiv A'$. Let φ be an **Xor***-derivation of A' from A. Then, $F \sqcup \Phi(\varphi)$ is a* **Rat***-derivation of $F \cup \mathcal{D}(A')$ from $F \cup \mathcal{D}(A)$.*

Proof. $\Phi(\varphi)$ is a derivation of $\mathcal{D}(A')$ from $\mathcal{D}(A)$ sharp in **TVar**. Since F is a basic CNF formula, $\mathrm{vars}(F)$ is disjoint from **TVar**, and thus the fusion $F \sqcup \Phi(\varphi)$ is a **Rat**-derivation as required in the claim. □

8.2.1 Generation of DRAT format proofs

Propositions 8.4 and 8.5 allow the generation of full unsatisfiability proofs for XOR-CDCL SAT solvers as **Rat**-derivations. Although this provides a whole generation procedure, it is rather impractical without one last feature: our approach is only capable to generate **Rat**-derivations, which are defined as strings of generally large CNF formulae. In Section 4.2.1, DRAT representations of **Rat**-derivations were introduced. Proposition 4.17 allows the construction of DRAT representations from **Rat**-derivations, but this constitutes no improvement if the full **Rat**-derivation still needs to be generated.

The utility of our notion of DRAT representations is that they provide the same information as represented **Rat**-derivations in a compact format. However, from a computational point of view, this is useless if the full **Rat**-derivation must be constructed prior to the generation of DRAT representations, because

whole formulae are recorded in **Rat**-derivations. In this Thesis, unsatisfiability proofs are constructed by composing **Rat**-derivations to obtain longer derivations, namely by composition, fusion with formulae and fusion of derivations. We present results that allow operating directly on DRAT representations of elementary **Rat**-derivations to obtain DRAT representations of composite **Rat**-derivations. In practice, this means that, when implementing the techniques presented in this Thesis, data structures for DRAT representations can directly be used instead of storing or operating **Rat**-derivations, which is typically unfeasible.

Definition 8.6 (Neglection of a CNF formula in a DRAT representation):
Let H be a CNF formula and $\varepsilon = [C_1^{x_1}, \ldots, C_n^{x_n}]$ be a string of labeled clauses. We define the neglection of H in ε as the string of clauses $H \triangleright \varepsilon = [C_1^{y_1}, \ldots, C_n^{y_n}]$ where $y_i = \mathsf{N}$ if $x_i = \mathsf{D}$ and $C_i \in H$, and $y_i = x_i$ otherwise, for every $1 \leq i \leq n$.

Example 8.7 (The role of N labels):
Consider clauses

$$C_1 = \mathsf{or}(\mathsf{p}_1, \mathsf{p}_2) \qquad C_2 = \mathsf{or}(\neg\mathsf{p}_1, \mathsf{p}_3) \qquad C_3 = \mathsf{or}(\mathsf{p}_2, \mathsf{p}_3) \qquad C_4 = \mathsf{or}(\neg\mathsf{p}_2, \mathsf{p}_4)$$

and CNF formulae

$$F_0 = \{C_1, C_2\} \qquad F_1 = \{C_1, C_2, C_3\} \qquad F_2 = \{C_1, C_3\} \qquad H = \{C_2, C_4\}$$

The string of CNF formulae $\alpha = [F_0, F_1, F_2]$ is a **At**-derivation of F_2 from F_0, therefore a **Rat**-derivation sharp in $\emptyset$. A DRAT representation of α sharp in $\emptyset$ is given by $\langle F_0, \varepsilon\rangle$ where $\varepsilon = [C_3^{\mathsf{I}}, C_2^{\mathsf{D}}]$. Now consider the **Rat**-derivation $H \sqcup \alpha = [H \cup F_0\ ,\ H \cup F_1\ ,\ H \cup F_2]$, and observe that the unions are given by:

$$H \cup F_1 = \{C_1, C_2, C_4\} \qquad H \cup F_2 = H \cup F_3 = \{C_1, C_2, C_3, C_4\}$$

$\langle F_0 \cup H, \varepsilon\rangle$ is not a DRAT representation of $H \sqcup \alpha$, since C_2 is labeled by D in ε and thus:

$$(H \cup F_0) * \varepsilon = [H \cup F_0\ ,\ H \cup F_1\ ,\ \{C_1, C_3, C_4\}]$$

However, $\langle H \cup F_0, H \triangleright \varepsilon\rangle$ is a DRAT representation of α with $H \triangleright \varepsilon = [C_3^{\mathsf{I}}, C_2^{\mathsf{N}}]$. ■

The notion of neglection allows to easily prove the following results, which are the last piece in our method. They provide simple ways to compute unsatisfiability proofs without having recourse to **Rat**-derivations. Instead, all proof composition operations in this Thesis can be directly performed over DRAT representations using these results.

Proposition 8.8 (DRAT representations of concatenated Rat-derivations):
Let α be a **Rat***-derivation of a CNF formula G from a CNF formula F sharp in a set of variables V; and β be a* **Rat***-derivation of a CNF formula H from G sharp in a set of variables U. Consider DRAT representations $\langle F, \varepsilon\rangle$ of α sharp in V, and $\langle G, \eta\rangle$ of β sharp in U. Then, $\varepsilon : \eta$ is a DRAT representation of $\alpha : \beta$ sharp in $V \cup U$.*
Proof. Assume that $\alpha = [F_0, \ldots, F_n]$ and $\beta = [F_{n+1}, \ldots, F_{n+r}]$ with $F = F_0$, $G = F_n = F_{n+1}$ and $H = F_{n+r}$. Since $\langle F, \varepsilon\rangle$ is a DRAT representation of α sharp in V, we can find strings of labeled clauses $\zeta_1, \ldots, \zeta_n$ such that:

- $\varepsilon = \zeta_1 : \ldots : \zeta_n$.
- For all $1 \leq i \leq n$, the string of CNF formulae $F_{i-1} * \zeta_i$ is a **Rat**-derivation of F_i from F_{i-1} sharp in V.

Similarly, because $\langle G, \eta\rangle$ is a DRAT representation of β sharp in U, we can find strings of labeled clauses $\zeta_{n+2}, \ldots, \zeta_{n+r}$ such that:

- $\eta = \zeta_{n+2} : \ldots : \zeta_{n+r}$.

- For all $n+2 \leq i \leq n+r$, the string of CNF formulae $F_{i-1} * \zeta_i$ is a **Rat**-derivation of F_i from F_{i-1} sharp in U.

Observe that in both cases, the **Rat**-derivations $F_{i-1} * \zeta_i$ are sharp in $V \cup U$. Consider now $\zeta_{n+1} = []$, whose derived string of CNF formulae is $F_n * \zeta_{n+1} = [F_n]$, a **Rat**-derivation of $F_n = F_{n+1}$ from F_n sharp in $V \cup U$. Furthermore, we have:

$$\zeta_1 : \ldots : \zeta_{n+r} = \zeta_1 : \ldots : \zeta_n : [] : \zeta_{n+2} : \ldots : \zeta n + r = \varepsilon : [] : \eta = \varepsilon : \eta$$

This shows that $\langle F, \varepsilon : \eta \rangle$ is a DRAT representation of $\alpha : \beta$ sharp in $V \cup U$. □

Proposition 8.9 (DRAT representations of fusions of CNF formulae into Rat-derivations):
Let $\alpha = [F_0, \ldots, F_n]$ be a **Rat***-derivation of F_n from F_0 sharp in a set of variables V, and H be a CNF formula such that* $\mathrm{vars}(H)$ *is disjoint from V. Let $\langle F_0, \varepsilon \rangle$ be a DRAT representation of α sharp in V. Then, $\langle H \cup F_0, H \triangleright \varepsilon \rangle$ is a DRAT representation of $H \sqcup \alpha$ sharp in V.*

Proof. The string $H \sqcup \alpha$, which is a **Rat**-derivation by Proposition 3.35, is given by $[H \cup F_0, \ldots, H \cup F_n]$. Consider a decomposition $\varepsilon = \varepsilon_1 : \ldots : \varepsilon_n$ as in Definition 4.15. It is straightforward to check that the equality $H \triangleright \varepsilon = (H \triangleright \varepsilon_1) : \ldots : (H \triangleright \varepsilon_n)$ holds. We now show that, for every $1 \leq i \leq n$, the string of CNF formulae $(H \cup F_{i-1}) * (H \triangleright \varepsilon_i)$ is a **Rat**-derivation of $H \cup F_i$ from $H \cup F_{i-1}$ sharp in V. Let us write:

$$\varepsilon_i = [C_1^{x_1}, \ldots, C_p^{x_p}] \qquad F_{i-1} * \varepsilon_i = [G_0, \ldots, G_p]$$
$$H \triangleright \varepsilon_i = [C_1^{y_1}, \ldots, C_p^{y_p}] \qquad (H \cup F_{i-1}) * (H \triangleright \varepsilon_i) = [G'_0, \ldots, G'_p]$$

We show by induction on j that $G'_j = H \cup G_j$ for all $0 \leq j \leq p$. The base case $j = 0$ is straightforward from Definition 4.14. Now assume that $G'_i = H \cup G_j$ for some $0 \leq j < p$, and let us show that $G'_{j+1} = H \cup G_{j+1}$. We distinguish four cases:

- If $x_{j+1} = \mathsf{I}$, then $y_{j+1} = \mathsf{I}$, and we obtain:

$$G'_{j+1} = G'_j \cup \{C_{j+1}\} = H \cup G_j \cup \{C_{j+1}\} = H \cup G_{j+1}$$

- If $x_{j+1} = \mathsf{N}$, then $y_{j+1} = \mathsf{N}$, and then $G'_{j+1} = G'_j = H \cup G_j = H \cup G_{j+1}$.
- If $x_{j+1} = \mathsf{D}$ and $C_{j+1} \notin H$, then $y_{j+1} = \mathsf{D}$, and we obtain:

$$G'_{j+1} = G'_j \setminus \{C_{j+1}\} = (H \cup G_j) \setminus \{C_{j+1}\} = H \cup (G_j \setminus \{C_{j+1}\}) = H \cup G_{j+1}$$

- If $x_{j+1} = \mathsf{D}$ and $C_{j+1} \in H$, then $y_{j+1} = \mathsf{N}$, and we obtain:

$$G'_{j+1} = G'_j = (H \cup G_j) = H \cup (G_j \setminus \{C_{j+1}\}) = H \cup G_{j+1}$$

By induction, $G'_j = H \cup G_j$ for all $0 \leq j \leq p$. This means that $(H \cup F_{i-1}) * (H \triangleright \varepsilon_i) = H \sqcup (F_{i-1} * \varepsilon_i)$. Since $F_{i-1} * \varepsilon_i$ is a **Rat**-derivation of F_i from F_{i-1} sharp in V and $\mathrm{vars}(H)$ is disjoint from V, then by Proposition 3.35 $H \sqcup (F_{i-1} * \varepsilon_i)$ is a **Rat**-derivation of $H \cup F_i$ from $H \cup F_{i-1}$ sharp in V, for every $1 \leq i \leq n$. Then, it follows that $\langle H \cup F_0, H \triangleright \varepsilon \rangle$ is a DRAT representation of $H \sqcup \alpha$ sharp in V. □

Proposition 8.10 (DRAT representations of fusions of Rat-derivations):
Let α_i be a **Rat***-derivation of a CNF formula G_i from another CNF formula F_i sharp in a set of variables V_i for every $1 \leq i \leq n$. Assume that, for every $1 \leq i, j \leq n$ with $i \neq j$, the set of variables* $\mathrm{vars}(F_j \cup G_j)$ *is disjoint from V_i. Let*

$\langle F_i, \varepsilon_i \rangle$ be a DRAT representation of α_i sharp in V_i for each $1 \leq i \leq n$, and define

$$\eta_i = \left(\bigcup_{j=1}^{i-1} F_j \cup \bigcup_{j=i+1}^{n} G_j \right) \triangleright \varepsilon_i$$

for every $1 \leq i \leq n$. Then, $\langle \bigcup_{i=1}^{n} F_i, \eta_1 : \ldots : \eta_n \rangle$ is a DRAT representation of $\alpha_1 \sqcup \cdots \sqcup \alpha_n$ sharp in $\bigcup_{i=1}^{n} V_i$.

Proof. By Definition 3.31, the fusion $\alpha_1 \sqcup \cdots \sqcup \alpha_n$ is a string of CNF formulae defined as the concatenation $\beta_1 : \ldots : \beta_n$, where

$$\beta_i = \left(\bigcup_{j=0}^{i-1} F_j \cup \bigcup_{j=i+1}^{n} G_j \right) \sqcup \alpha_i$$

for every $1 \leq i \leq n$. By Proposition 3.35, for every $1 \leq i \leq n$, the string of CNF formulae β_i is a **Rat**-derivation of $\left(\bigcup_{j=0}^{i-1} F_j \cup \bigcup_{j=i}^{n} G_j\right)$ from $\left(\bigcup_{j=0}^{i} F_j \cup \bigcup_{j=i+1}^{n} G_j\right)$ sharp in V_i. Proposition 8.10 then shows that

$$\left\langle \left(\bigcup_{j=0}^{i} F_j \cup \bigcup_{j=i+1}^{n} G_j \right), \eta_i \right\rangle$$

is a DRAT representation of β_i sharp in V_i.

Now, Proposition 3.34 implies that $\beta_1 : \ldots : \beta_n$ is a **Rat**-derivation of $\bigcup_{i=1}^{n} G_i$ from $\bigcup_{i=1}^{n} F_i$ sharp in $\bigcup_{i=1}^{n} V_i$. Hence, applying Proposition 8.8 shows that $\langle \bigcup_{i=1}^{n} F_i, \eta_1 : \ldots : \eta_n \rangle$ is a DRAT representation of $\beta_1 : \ldots : \beta_n$ sharp in $\bigcup_{i=1}^{n} V_i$, as we wanted to show. □

8.3 Contributions

In this chapter, the compared lengths of direct and linear translations have been studied. In Theorem 8.2, we find that linear translations are polynomially bounded under reasonable bounds for the size of the original **Xor**-derivation. The worst-case length of direct translations is hugely larger: Example 8.3 shows a family of affine formuale on which direct translations of executions of Gauss-Jordan elimination are exponentially sized. This shows a strong separation between direct and linear translations.

However, this is only a worst-case, asymptotical result. In Section 8.1.2, we find that, in some practical situations, direct translations overperform linear translations, specifically whenever the ω-measure along the original **Xor**-derivation remains small. Nevertheless, linear translations are much shorter in the rest of the cases. Hence, we propose the use of the bounds computed along Chapters 6 and 7 to predict the shortest method prior to generation of a translation.

The integration of our methods in the general framework for unsatisfiability proof generation from Section 4.2.2 is presented in Section 8.2.1. In particular, proof generation for the rules $\xrightarrow[\text{X-simp}]{}$ and $\xrightarrow[\text{X-learn}]{}$ is presented in Propositions 8.5 and 8.4 respectively, which yields a complete unsatisfiability proof generation method for XOR-CDCL SAT solvers.

Applying directly this method is unfeasible, because **Rat**-derivations must be generated before converting them to their DRAT representations. Propositions 8.8, 8.9 and 8.10 provide techniques that allow the generation of DRAT unsatisfiability proofs without previously generating **Rat**-derivations. This is done by translating our composition operations on **Rat**-derivations, namely concatenation, fusion with formulae and fusion of derivations, into composition operations on DRAT representations.

Chapter 9

Conclusions

The advent of SAT solvers with integrated parity reasoning procedures allowed a successful application of SAT solving to problems with encoded affine formulae. This is particularly relevant in the field of cryptography, where classical CDCL SAT solvers showed a poor performance. However, the integration of parity reasoning within SAT solvers posed a problem whenever unsatisfiability proofs must be issued, since no generation method for parity reasoning techniques was known. Therefore, state-of-the-art solvers are currently forced to disable parity reasoning under such requirements.

A separate issue regarding unsatisfiability proof generation is the lack of an extensive formalization of proof systems that includes the DRAT proof standard as a naturally defined system. Methods for unsatisfiability proof generation are presented in the literature in a rather loose way. However, the idiosyncrasy of DRAT proofs, particularly its lack of soundness with respect to the entailment relation and its context-dependent behavior (its non-deepness, in the language of this Thesis), makes a formalization necessary to provide additional reliability. This is capital to guarantee correctness of results reported by SAT solvers.

Throughout this Thesis, these problems have been tackled and several contributions have been proposed, which together allow a solution to the problem of generating unsatisfiability proofs for SAT solvers when parity reasoning techniques are used. An outline of the obtained methods and frameworks follows.

A formal framework for proof systems. A novel notion of proof system has been presented in Chapter 3. The main advantage of our framework is that it allows an unified perspective on most proof systems relevant for SAT solvers coupled with specialized reasoning procedure. The DRAT proof format is correctly modeled within this framework through the concept of **Rat**-derivation. The use of whole formulae in the notion of derivation, in contrast to the standard conception of proofs as sequences of constraints, allows to get rid of problems in proof composition for non-deep proof systems.

Besides its use to formalize proofs in classical environments as CNF formulae, our framework allows the definition of proof systems over arbitrary logical languages. In particular, we have been able to define the XOR proof system, which models the Gauss-Jordan elimination procedure as a theorem proving procedure with an underlying proof system. Furthermore, since non-deep proof systems are now natural in our framework, we are able to introduce an extended resolution-like inference schema in the XOR proof system, which yields the extended XOR proof system. In the same way that resolution proofs can be shrinked by the use of definitions [CR79], extended XOR variants of XOR proofs can be obtained with a much better asymptotical behavior, which are given by the intermediate derivations presented in Section 7.2.

A generalized schema for clause learning. One of the crucial properties of the XOR-CDCL approach for SAT solving with integrated parity reasoning is that applications of the classical CDCL rules and the parity reasoning rules may be interleaved. This is problematic for unsatisfiability proof generation, since the

classical approach relies on learned clauses being linear resolvents on the current CNF formula. However, if literals are propagated using parity reasoning, this condition no longer holds. In particular, clauses learned by the $\xrightarrow[\text{X-learn}]{}$ rule for parity reasoning-based clause learning are not learned by any application of the $\xrightarrow[\text{C-learn}]{}$ rule for classical clause learning.

In order to provide an unified framework for unsatisfiability proof generation, we devise a general setting for clause learning in Section 4.1, which can be instantiated for different conflict analysis techniques used in a solver. Through the use of this framework, we provide an unsatisfiability proof schema where a full unsatisfiability proof is generated from derivations of individual reason clauses. This framework correctly models both classical and parity-based conflict analysis and clause learning.

Translations through the direct encoding. The unsatisfiability proof generation method described above requires to derive reason clauses and simplified formulae. We are able to show in Section that this boils down to provide translations of **Xor**-derivations describing the execution of a Gauss-Jordan elimination procedure into **Rat**-derivations of the encodings of derived affine formulae. The first method to obtain such translations, explained in Chapter 6 is called the direct translation. Although the direct translation fulfills the conditions required for a translation to be used for unsatisfiability proof generation, derivations experience an exponential blow up when they are translated.

The second method is explained in Chapter 7, and is called linear translation. Linear translations exploit the aforementioned reduction of size using the extended XOR proof system; extended XOR derivations, called intermediate derivations, can then be applied the direct translation to obtain a lift derivation. The lift derivation is only able to derive the linear encoding of conclusions of the original XOR derivation from the linear encoding of its premises. Therefore, derivations converting direct encodings to linear encodings and vice versa must be appended to the lift.

As discussed in Section 8.1, linear translations are, from a worst-case complexity perspective, exponentially shorter than direct translations. Nonetheless, in practical situations where the original XOR proof does not derive large parity constraints, direct translations perform notably better than linear translations. Hence, we propose to choose between linear or direct translations in runtime, either by computing precise approximations of translation lengths, or by a heuristic based on a measure that empirically shows a strong relation with the relative performance of direct and linear translations.

9.1 Future work

Although this Thesis provides a complete method for unsatisfiability proof generation for XOR-CDCL SAT solvers, several improvements and extensions are possible.

Implementation issues. The proposed methods have been described in full detail. Still, no practical implementation has been developed. Doing so would be useful to find unexpected shortcomings of our approach that are not theoretically apparent.

One aspect where our methods can be improved is our choice of regularized splittings instead of natural splittings [GK13] in the linear encoding. The natural splitting of a parity constraint is a reduced version of the regularized splitting where extremal parity constraints are modified. The irregular form of natural splittings involves distinguishing many cases in the definition of the correspondent linear translations. However, we do not find a reason why an extension of the linear translation where natural splittings are used in the intermediate translation cannot be constructed by discussing all cases. An immediate reason for such an extension would be beneficial is that small parity constraints are represented by a single parity constraint in natural splittings, which is likely to be the cause of the poor performance of linear translations when the ω measure along the original **Xor**-derivation remains low.

Further optimization may be provided by preserving linear encodings of premises derived by prefix derivations instead of deleting them. This would allow to avoid repeatedly inserting prefixes in translations, thus avoiding to some extent the exponentially-sized part of linear translations.

Last, our methods work not only for **Xor**-derivations that are generated by Gauss-Jordan elimination, but for arbitrary ones. Refining the original **Xor**-derivation into another one with lower translation length might be feasible. However, since this would only modify the length of the lift in the linear translation, which has linear length in the length of the original **Xor**-derivation, a large improvement cannot not expected.

Extension to different proof systems. The most promising extension to our work would be to enable unsatisfiability proof generation for SAT solvers with integrated cardinality reasoning, which is still an open problem [HB15]. Since integrated cardinality reasoning techniques are based on the cutting-planes proof system, our approach can be applied by providing suitable translations. Indeed, this was one of the main goals behind the development of general frameworks in our work. By introducing translations of cutting-planes derivations into **Rat**-derivations, similar results could be obtained.

Last, from a purely theoretical point of view, it would be interesting to find how far our framework for proof systems can get, and whether there are general use proof systems that cannot be modeled by our framework. Furthermore, since context-dependent proof systems are allowed, non-deep extensions to known proof systems can now be proposed and studied from a structural point of view, which is a shortcoming in current structural approaches to proof systems [Bru15]. For example, our methods allow the study of the extended resolution [Tse83] and extended Frege [Bus15] proof systems.

Bibliography

[ARMS03] Fadi A. Aloul, Arathi Ramani, Igor L. Markov, and Karem A. Sakallah. Solving difficult instances of boolean satisfiability in the presence of symmetry. *IEEE Transactions on CAD of Integrated Circuits and Systems*, 22(9):1117–1137, 2003.

[Arn09] Holger Arnold. A linearized DPLL calculus with clause learning (2nd, revised version). Technical report, 2009.

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

[AS14] Gilles Audemard and Laurent Simon. Glucose in the SAT 2014 competition. In Anton Belov, Daniel Diepold, Marijn Heule, and Matti Järvisalo, editors, *Proceedings of SAT Competition 2014. Solver and Benchmark Descriptions*, volume B-2014-2 of *Department of Computer Science Series of Publications B*, pages 31–32. University of Helsinki, 2014.

[BBLM14] Armin Biere, Daniel Le Berre, Emmanuel Lonca, and Norbert Manthey. Detecting cardinality constraints in CNF. In Carsten Sinz and Uwe Egly, editors, *17th International Conference on Theory and Applications of Satisfiability Testing — SAT 2014*, volume 8561 of *Lecture Notes in Computer Science*, pages 285–301. Springer, 2014.

[BBR06] Olivier Bailleux, Yacine Boufkhad, and Olivier Roussel. A translation of pseudo boolean constraints to SAT. *JSAT*, 2(1-4):191–200, 2006.

[BDHJ14] Anton Belov, Daniel Diepold, Marijn Heule, and Matti Järvisalo, editors. *Proceedings of SAT Competition 2014. Solver and Benchmark Descriptions*, volume B-2014-2 of *Department of Computer Science Series of Publications B*. University of Helsinki, 2014.

[Bes06] Christian Bessiere. Constraint propagation. In Francesca Rossi, Peter van Beek, and Toby Walsh, editors, *Handbook of Constraint Programming*, chapter 3, pages 27–81. Elsevier, 2006.

[BGW94] Leo Bachmair, Harald Ganzinger, and Uwe Waldmann. Refutational theorem proving for hierarchic first-order theories. *Applicable Algebra in Engineering, Communication and Computing*, 5:193–212, 1994.

[BHM14] Anton Belov, Marijn Heule, and João Marques-Silva. MUS extraction using clausal proofs. In Carsten Sinz and Uwe Egly, editors, *17th International Conference on Theory and Applications of Satisfiability Testing — SAT 2014*, volume 8561 of *Lecture Notes in Computer Science*, pages 48–57. Springer, 2014.

[BHvMW09] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[Bie14] Armin Biere. Yet another local search solver and Lingeling and friends entering the SAT competition 2014. In Anton Belov, Daniel Diepold, Marijn Heule, and Matti Järvisalo, editors, *Proceedings of SAT Competition 2014. Solver and Benchmark Descriptions*, volume B-2014-2 of *Department of Computer Science Series of Publications B*, pages 31–32. University of Helsinki, 2014.

[BK95] Manuel Blum and Sampath Kannan. Designing programs that check their work. *Journal of the ACM*, 42(1):269–291, 1995.

[BKS04] Paul Beame, Henry A. Kautz, and Ashish Sabharwal. Towards understanding and harnessing the potential of clause learning. *Journal of Artificial Intelligence Research (JAIR)*, 22:319–351, 2004.

[BLB10] Robert Brummayer, Florian Lonsing, and Armin Biere. Automated testing and debugging of SAT and QBF solvers. In Ofer Strichman and Stefan Szeider, editors, *Theory and Applications of Satisfiability Testing - SAT 2010, 13th International Conference, SAT 2010, Edinburgh, UK, July 11-14, 2010. Proceedings*, volume 6175 of *Lecture Notes in Computer Science*, pages 44–57. Springer, 2010.

[BM00] Peter Baumgartner and Fabio Massacci. The taming of the (X)OR. In John W. Lloyd, Verónica Dahl, Ulrich Furbach, Manfred Kerber, Kung-Kiu Lau, Catuscia Palamidessi, Luís Moniz Pereira, Yehoshua Sagiv, and Peter J. Stuckey, editors, *Proceedings of the First International Conference on Computational Logic — CL 2000*, volume 1861 of *Lecture Notes in Computer Science*, pages 508–522. Springer, 2000.

[Bru15] Paola Bruscoli. Personal communication, 2015.

[Bry86] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986.

[Bus98] Samuel R. Buss. An introduction to proof theory. In Samuel R. Buss, editor, *Handbook of proof theory*, Studies in logic and the foundations of mathematics, chapter 1. Elsevier, 1998.

[Bus15] Sam Buss. Propositional proofs in frege and extended frege systems (abstract). In Lev D. Beklemishev and Daniil V. Musatov, editors, *10th International Computer Science Symposium in Russia — CSR 2015*, volume 9139 of *Lecture Notes in Computer Science*, pages 1–6. Springer, 2015.

[CB07] Nicolas Courtois and Gregory V. Bard. Algebraic cryptanalysis of the data encryption standard. In Steven D. Galbraith, editor, *11th IMA International Conference on Cryptography and Coding*, volume 4887 of *Lecture Notes in Computer Science*, pages 152–169. Springer, 2007.

[CCT87] William Cook, Collette R. Coullard, and György Turán. On the complexity of cutting-plane proofs. *Discrete Applied Mathematics*, 18(1):25–38, 1987.

[Che09] Jingchao Chen. Building a hybrid SAT solver via conflict-driven, look-ahead and XOR reasoning techniques. In Oliver Kullmann, editor, *12th International Conference on Theory and Applications of Satisfiability Testing — SAT 2009*, volume 5584 of *Lecture Notes in Computer Science*, pages 298–311. Springer, 2009.

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In Michael A. Harrison, Ranan B. Banerji, and Jeffrey D. Ullman, editors, *3rd Annual ACM Symposium on Theory of Computing*, pages 151–158. Association for Computing Machinery, 1971.

[CR79] Stephen A. Cook and Robert A. Reckhow. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44(1):36–50, 1979.

[Day07] Jane Day. Vectors, matrices, and systems of linear equations. In Leslie Hogben, editor, *Handbook of Linear Algebra*, Discrete Mathematics and its Applications, chapter 1. CRC Press, 2007.

[Dia06] Razvan Diaconescu. Proof systems for institutional logic. *Journal of Logic and Computation*, 16(3):339–357, 2006.

[DLL62] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

[EB05] Niklas Eén and Armin Biere. Effective preprocessing in SAT through variable and clause elimination. In Fahiem Bacchus and Toby Walsh, editors, *8th International Conference on Theory and Applications of Satisfiability Testing — SAT 2005*, volume 3569 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 2005.

[ES06] Niklas Eén and Niklas Sörensson. Translating pseudo-boolean constraints into SAT. *JSAT*, 2(1-4):1–26, 2006.

[GB92] Joseph A. Goguen and Rod M. Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the ACM*, 39(1):95–146, 1992.

[GGW+03] Aarti Gupta, Malay K. Ganai, Chao Wang, Zijiang Yang, and Pranav Ashar. Learning from bdds in sat-based bounded model checking. In *40th Design Automation Conference — DAC 2003*, pages 824–829. Association for Computing Machinery, 2003.

[GHM+12] Peter Großmann, Steffen Hölldobler, Norbert Manthey, Karl Nachtigall, Jens Opitz, and Peter Steinke. Solving periodic event scheduling problems with SAT. In He Jiang, Wei Ding, Moonis Ali, and Xindong Wu, editors, *25th International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems — IEA/AIE 2012*, volume 7345 of *Lecture Notes in Computer Science*, pages 166–175. Springer, 2012.

[GJ79] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[GK13] Matthew Gwynne and Oliver Kullmann. On SAT representations of XOR constraints. *Computing Research Repository*, abs/1309.3060, 2013.

[GN03] Evguenii I. Goldberg and Yakov Novikov. Verification of proofs of unsatisfiability for CNF formulas. In *2003 Design, Automation and Test in Europe Conference and Exposition — DATE 2003*, pages 10886–10891. IEEE Computer Society, 2003.

[Gol08] Odred Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, 2008.

[GSK98] Carla P. Gomes, Bart Selman, and Henry A. Kautz. Boosting combinatorial search through randomization. In Jack Mostow and Chuck Rich, editors, *15th National Conference on Artificial Intelligence and 10th Conference on on Innovative Applications of Artificial Intelligence*, pages 431–437. AAAI Press / The MIT Press, 1998.

[Gug99] Alessio Guglielmi. A system of interaction and structure. *Computing Research Repository*, cs.LO/9910023, 1999.

[Hak85] Armin Haken. The intractability of resolution. *Theoretical Computer Science*, 39:297–308, 1985.

[Hal76] Paul Halmos. *Measure Theory*. Graduate Texts in Mathematics. Springer New York, 1976.

[HB15] Marijn Heule and Armin Biere. Proofs for satisfiability problems. In Bruno Woltzenlogel Paleo and David Delahaye, editors, *All about Proofs, Proofs for all*, volume 55 of *Mathematical logic and foundations*. College Publications, 2015.

[HDvZvM04] Marijn Heule, Mark Dufour, Joris van Zwieten, and Hans van Maaren. March_eq: Implementing additional reasoning into an efficient look-ahead SAT solver. In Holger H. Hoos and David G. Mitchell, editors, *7th International Conference on Theory and Applications of Satisfiability Testing — SAT 2004*, volume 3542 of *Lecture Notes in Computer Science*, pages 345–359. Springer, 2004.

[Heu08] Marijn Heule. *SmArT solving: Tools and techniques for satisfiability solvers*. PhD thesis, TU Delft, 2008.

[Heu14] Marijn Heule. march: Towards a lookahead SAT solver for general purposes. Master's thesis, TU Delft, 2014.

[HJ12] Cheng-Shen Han and Jie-Hong Roland Jiang. When boolean satisfiability meets gaussian elimination in a simplex way. In P. Madhusudan and Sanjit A. Seshia, editors, *24th International Conference on Computer Aided Verification — CAV 2012*, pages 410–426, 2012.

[HJB10] Marijn Heule, Matti Järvisalo, and Armin Biere. Clause elimination procedures for CNF formulas. In Christian G. Fermüller and Andrei Voronkov, editors, *17th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning — LPAR-17*, volume 6397 of *Lecture Notes in Computer Science*, pages 357–371. Springer, 2010.

[HJB13] Marijn Heule, Matti Järvisalo, and Armin Biere. Revisiting hyper binary resolution. In Carla P. Gomes and Meinolf Sellmann, editors, *10th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems — CPAIOR 2013*, volume 7874 of *Lecture Notes in Computer Science*, pages 77–93. Springer, 2013.

[HJW14] Marijn Heule, Warren A. Hunt Jr., and Nathan Wetzler. Bridging the gap between easy generation and efficient verification of unsatisfiability proofs. *Software Testing Verification and Reliability*, 24(8):593–607, 2014.

[HMPS14] Steffen Hölldobler, Norbert Manthey, Tobias Philipp, and Peter Steinke. Generic CDCL - A formalization of modern propositional satisfiability solvers. In Daniel Le Berre, editor, *5th Workshop on Pragmatics of SAT — POS 2014*, volume 27 of *EPiC Series*, pages 89–102. EasyChair, 2014.

[Hoo88] John N. Hooker. Generalized resolution and cutting planes. *Annals of Operation Research*, 12(1-4):217–239, 1988.

[HS09] HyoJung Han and Fabio Somenzi. On-the-fly clause improvement. In Oliver Kullmann, editor, *12th International ConferenceTheory and Applications of Satisfiability Testing — SAT 2009*, volume 5584 of *Lecture Notes in Computer Science*, pages 209–222. Springer, 2009.

[HvM05] Marijn Heule and Hans van Maaren. Aligning CNF- and equivalence-reasoning. In Holger H. Hoos and David G. Mitchell, editors, *7th International Conference on Theory and Applications of Satisfiability Testing — SAT 2004*, volume 3542 of *Lecture Notes in Computer Science*, pages 145–156. Springer, 2005.

[JHB12] Matti Järvisalo, Marijn Heule, and Armin Biere. Inprocessing rules. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *6th International Joint Conference on Automated Reasoning — IJCAR 2012*, volume 7364 of *Lecture Notes in Computer Science*, pages 355–370. Springer, 2012.

[JS04] HoonSang Jin and Fabio Somenzi. Circus: A hybrid satisfiability solver. In Holger H. Hoos and David G. Mitchell, editors, *7th International ConferenceTheory and Applications of Satisfiability Testing — SAT 2004*, volume 3542 of *Lecture Notes in Computer Science*, pages 211–223. Springer, 2004.

[JSB06] Toni Jussila, Carsten Sinz, and Armin Biere. Extended resolution proofs for symbolic SAT solving with quantification. In Armin Biere and Carla P. Gomes, editors, *9th International Conference on Theory and Applications of Satisfiability Testing - SAT 2006*, volume 4121 of *Lecture Notes in Computer Science*, pages 54–60. Springer, 2006.

[KL99] Hans Kleine Büning and Theodor Lettmann. *Propositional Logic: Deduction and Algorithms*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1999.

[KS92] Henry A. Kautz and Bart Selman. Planning as satisfiability. In Bernd Neumann, editor, *10th European Conference on Artificial Intelligence — ECAI 92*, pages 359–363, 1992.

[Lai14] Tero Laitinen. *Extending SAT Solver with Parity Reasoning*. PhD thesis, Aalto University, 2014.

[Li00] Chu Min Li. Integrating equivalency reasoning into davis-putnam procedure. In Henry A. Kautz and Bruce W. Porter, editors, *17th National Conference on Artificial Intelligence and 12th Conference on on Innovative Applications of Artificial Intelligence*, pages 291–296. AAAI Press / The MIT Press, 2000.

[Li03] Chu Min Li. Equivalent literal propagation in the DLL procedure. *Discrete Applied Mathematics*, 130(2):251–276, 2003.

[LJN10] Tero Laitinen, Tommi Junttila, and Ilkka Niemelä. Extending clause learning DPLL with parity reasoning. In *Proceedings of the 19th European Conference on Artificial Intelligence — ECAI 2010*, pages 21–26. IOS Press, 2010.

[LJN11] Tero Laitinen, Tommi Junttila, and Ilkka Niemelä. Equivalence class based parity reasoning with DPLL(XOR). In *IEEE 23rd International Conference on Tools with Artificial Intelligence — ICTAI 2011*, pages 649–658. IEEE Computer Society, 2011.

[LJN12] Tero Laitinen, Tommi Junttila, and Ilkka Niemelä. Extending clause learning SAT solvers with complete parity reasoning (extended version). *Computing Research Repository*, abs/1207.0988, 2012.

[LJN13] Tero Laitinen, Tommi A. Junttila, and Ilkka Niemelä. Simulating parity reasoning (extended version). *Computing Research Repository*, abs/1311.4289, 2013.

[LJN14a] Tero Laitinen, Tommi Junttila, and Ilkka Niemelä. Classifying and propagating parity constraints (extended version). *Computing Research Repository*, abs/1406.4698, 2014.

[LJN14b] Tero Laitinen, Tommi Junttila, and Ilkka Niemelä. Conflict-driven XOR-clause learning (extended version). *Computing Research Repository*, abs/1407.6571, 2014.

[LM06] Inês Lynce and João Marques-Silva. Efficient haplotype inference with boolean satisfiability. In *20th National Conference on Artificial Intelligence and 18th Conference on on Innovative Applications of Artificial Intelligence*, pages 104–109. AAAI Press, 2006.

[Man11] Norbert Manthey. Coprocessor - a standalone SAT preprocessor. *Computing Research Repository*, abs/1108.6208, 2011.

[Man14a] Norbert Manthey. Riss 4.27. In Anton Belov, Daniel Diepold, Marijn Heule, and Matti Järvisalo, editors, *Proceedings of SAT Competition 2014. Solver and Benchmark Descriptions*, volume B-2014-2 of *Department of Computer Science Series of Publications B*, pages 65–67. University of Helsinki, 2014.

[Man14b] Norbert Manthey. *Towards Next Generation Sequential and Parallel SAT Solvers*. PhD thesis, TU Dresden, 2014.

[Mar09] Filip Marić. Formalization and implementation of modern SAT solvers. *Journal of Automated Reasoning*, 43(1):81–119, 2009.

[MM00] Fabio Massacci and Laura Marraro. Logical cryptanalysis as a SAT problem. *Journal of Automated Reasoning*, 24(1/2):165–203, 2000.

[MMZ+01] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *38th Design Automation Conference — DAC 2001*, pages 530–535. Association for Computing Machinery, 2001.

[MMZM01] Matthew W. Moskewicz, Conor F. Madigan, Lintao Zhang, and Sharad Malik. Efficient conflict driven learning in a boolean satisfiability solver. In *2001 IEEE/ACM International Conference on Computer Aided Design — ICCAD 2001*, pages 279–285, 2001.

[Mot36] Theodore Samuel Motzkin. *Contributions to the Theory of Linear Inequalities*. PhD thesis, University of Basel, 1936.

[MP14] Norbert Manthey and Tobias Philipp. Formula simplifications as DRAT derivations. In Carsten Lutz and Michael Thielscher, editors, *37th Annual German Conference on Artificial Intelligence — KI 2014*, volume 8736 of *Lecture Notes in Computer Science*, pages 111–122. Springer, 2014.

[MPS14] Norbert Manthey, Tobias Philipp, and Peter Steinke. A more compact translation of pseudo-boolean constraints into CNF such that generalized arc consistency is maintained. In Carsten Lutz and Michael Thielscher, editors, *37th Annual German Conference on AI — KI 2014*, volume 8736 of *Lecture Notes in Computer Science*, pages 123–134. Springer, 2014.

[NOT05] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Abstract DPLL and abstract DPLL modulo theories. In Franz Baader and Andrei Voronkov, editors, *11th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning — LPAR 2004*, volume 3452 of *Lecture Notes in Computer Science*, pages 36–50. Springer, 2005.

[OSOC12] Duckki Oe, Aaron Stump, Corey Oliver, and Kevin Clancy. versat: A verified modern sat solver. In Viktor Kuncak and Andrey Rybalchenko, editors, *Verification, Model Checking, and Abstract Interpretation*, volume 7148 of *Lecture Notes in Computer Science*, pages 363–378. Springer Berlin Heidelberg, 2012.

[Pap07] Christos H. Papadimitriou. *Computational complexity*. Academic Internet Publications, 2007.

[PD07] Knot Pipatsrisawat and Adnan Darwiche. A lightweight component caching scheme for satisfiability solvers. In *10th International Conference on Theory and Applications of Satisfiability Testing — SAT 2007*, volume 4501 of *Lecture Notes in Computer Science*, pages 294–299. Springer, 2007.

[PD11] Knot Pipatsrisawat and Adnan Darwiche. On the power of clause-learning SAT solvers as resolution engines. *Artificial Intelligence*, 175(2):512–525, 2011.

[Pra02] Terry Pratchett. *Night Watch*. Doubleday, 2002.

[Pre09] Steven Prestwich. Cnf encodings. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, chapter 2, pages 75–97. IOS Press, 2009.

[Rin09] Jussi Rintanen. Planning and sat. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, chapter 15, pages 483–504. IOS Press, 2009.

[RM09] Olivier Roussel and Vasco Manquinho. Pseudo-boolean and cardinality constraints. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, chapter 22, pages 695–733. IOS Press, 2009.

[RP15] Adrián Rebola-Pardo. Refutationally complete hierarchic theorem proving with definitions. Master Project, 2015.

[Sak09] Karem A. Sakallah. Symmetry and satisfiability. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, chapter 10, pages 289–338. IOS Press, 2009.

[SB06] Carsten Sinz and Armin Biere. Extended resolution proofs for conjoining bdds. In Dima Grigoriev, John Harrison, and Edward A. Hirsch, editors, *1st International Computer Science Symposium in Russia — CSR 2006*, volume 3967 of *Lecture Notes in Computer Science*, pages 600–611. Springer, 2006.

[SHvM09] Bas Schaafsma, Marijn Heule, and Hans van Maaren. Dynamic symmetry breaking by simulating zykov contraction. In Oliver Kullmann, editor, *12th International Conference on Theory and Applications of Satisfiability Testing — SAT 2009*, volume 5584 of *Lecture Notes in Computer Science*, pages 223–236. Springer, 2009.

[Sip97] Michael Sipser. *Introduction to the theory of computation*. PWS Publishing Company, 1997.

[SLM09] João P. Marques Silva, Inês Lynce, and Sharad Malik. Conflict-driven clause learning SAT solvers. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, chapter 4, pages 131–153. IOS Press, 2009.

[SNC09] Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending SAT solvers to cryptographic problems. In Oliver Kullmann, editor, *12th International Conference on Theory and Applications of Satisfiability Testing — SAT 2009*, volume 5584 of *Lecture Notes in Computer Science*, pages 244–257. Springer, 2009.

[Soo12] Mate Soos. Enhanced gaussian elimination in DPLL-based SAT solvers. In Daniel Le Berre, editor, *5th Workshop on Pragmatics of SAT — POS 2010*, volume 8 of *EPiC Series*, pages 2–14. EasyChair, 2012.

[Soo14] Mate Soos. CryptoMiniSat v4. In Anton Belov, Daniel Diepold, Marijn Heule, and Matti Järvisalo, editors, *Proceedings of SAT Competition 2014. Solver and Benchmark Descriptions*, volume B-2014-2 of *Department of Computer Science Series of Publications B*, page 23. University of Helsinki, 2014.

[SS06] Hossein M. Sheini and Karem A. Sakallah. Pueblo: A hybrid pseudo-boolean SAT solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 2(1-4):165–189, 2006.

[Tse83] G. S. Tseitin. On the complexity of derivation in propositional calculus. In J. Siekmann and G. Wrightson, editors, *Automation of Reasoning 2: Classical Papers on Computational Logic 1967-1970*, pages 466–483. Springer, 1983.

[Urq87] Alasdair Urquhart. Hard examples for resolution. *Journal of the ACM*, 34(1):209–219, 1987.

[vD94] Dirk van Dalen. *Logic and Structure*. Universitext. Springer, 1994.

[WHJ13] Nathan Wetzler, Marijn Heule, and Warren A. Hunt Jr. Mechanical verification of SAT refutations with extended resolution. In Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie, editors, *4th International Conference on Interactive Theorem Proving — ITP 2013*, volume 7998 of *Lecture Notes in Computer Science*, pages 229–244. Springer, 2013.

[WHJ14] Nathan Wetzler, Marijn Heule, and Warren A. Hunt Jr. DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In Carsten Sinz and Uwe Egly, editors, *17th International Conference on Theory and Applications of Satisfiability Testing — SAT 2014*, volume 8561 of *Lecture Notes in Computer Science*, pages 422–429. Springer, 2014.

[WvM98] Joost P. Warners and Hans van Maaren. A two-phase algorithm for solving a class of hard satisfiability problems. *Operations Research Letters*, 23(3-5):81–88, 1998.

[WWLH14] Cheng-Yin Wu, Chi-An Wu, Chien-Yu Lai, and Chung-Yang R. Haung. A counterexample-guided interpolant generation algorithm for SAT-based model checking. *IEEE Transactions on CAD of Integrated Circuits and Systems*, 33(12):1846–1858, 2014.

[ZM03] Lintao Zhang and Sharad Malik. Validating SAT solvers using an independent resolution-based checker: Practical implementations and other applications. In *2003 Design, Automation and Test in Europe Conference and Exposition — DATE 2003*, pages 10880–10885. IEEE Computer Society, 2003.