UNIVERSITÉ D'ARTOIS

Extensions de classes polynomiales pour le problème de satisfaisabilité

THÈSE

présentée et soutenue publiquement le 14/11/2016
en vue de l'obtention du

Doctorat de l'Université d'Artois
(Spécialité Informatique)

par

MOHAMMAD SALEH BALASIM AL-SAEDI

Membres du Jury :

Rapporteurs:	Bélaïd Benhamou (Université de Aix-Marseille)	
	Chu Min Li (Université de Picardie Jules Verne)	
Examinateurs:	Laure Brisoux (Université de Picardie Jules Verne)	
	Éric Grégoire (Université d'Artois - France)	*Directeur de thèse*
	Bertrand Mazure (Université d'Artois - France)	*Co-directeur de thèse*
	Lakhdar Saïs (Université d'Artois - France)	*Co-directeur de thèse*

CENTRE DE RECHERCHE EN INFORMATIQUE DE LENS (CRIL CNRS UMR 8188)
Université d'Artois, rue Jean Souvraz, SP 18, F-62307, Lens Cedex France Secrétariat : Tél.: +33 (0)3 21 79 17 23 – Fax : +33 (0)3 21 79 17 70
http://www.cril.univ-artois.fr

UNIVERSITÉ D'ARTOIS

Extensions of Tractable Classes for Propositional Satisfiability

Ph. D. Thesis In Computer Science

University of Artois

publicly defended on 14/11/2016
by

MOHAMMAD SALEH BALASIM AL-SAEDI

Jury:

Reviewers:	Bélaïd Benhamou (Université de Aix-Marseille)	
	Chu Min Li (Université de Picardie Jules Verne)	
Examinators:	Laure Brisoux (Université de Picardie Jules Verne)	
	Éric Grégoire (Université d'Artois - France)	*Thesis supervisor*
	Bertrand Mazure (Université d'Artois - France)	*Thesis co-supervisor*
	Lakhdar SAÏS (Université d'Artois - France)	*Thesis co-supervisor*

CENTRE DE RECHERCHE EN INFORMATIQUE DE LENS (CRIL CNRS UMR 8188)
Université d'Artois, rue Jean Souvraz, SP 18, F-62307, Lens Cedex France Secrétariat : Tél.: +33 (0)3 21 79 17 23 – Fax : +33 (0)3 21 79 17 70
http://www.cril.univ-artois.fr

Contents

Part I

General Introduction

This dissertation thesis deals with propositional satisfiability (SAT), one of the most important problems in computing science, including artificial intelligence (A.I.) and complexity theory. Two main lines of research have been mainly investigated over the years in this domain. The first one deals with the design of efficient SAT solvers, allowing the classes of instances that can be solved in practice to be extended. The second one is mainly theoretically-oriented: the goal is to characterize new tractable classes or formulas that can be recognized and solved in polynomial time. In this dissertation thesis, the focus is on this second important issue, while keeping in mind the potential impact of our results in practice.

Propositional reasoning and search has indeed been a very active topic of research in A.I. the last three decades. Thanks to impressive improvements in the time-efficiency of satisfiability checking procedures on many instances (see for example [7]), the propositional satisfiability framework is now widely recognized as a powerful practical setting for many reasoning and A.I. problem-solving paradigms. SAT, namely checking whether a set of propositional clauses is satisfiable, has also attracted much attention for a very long time in theoretical computing science since it is a canonical NP-complete problem [8]; as such, SAT is thus expected to remain intractable in the worst case unless P=NP.

Despite the considerable research efforts over the years, theoretical results about various tractable fragments of SAT still hardly play a role in the implementation of the most efficient current SAT solvers. We believe that the extension of current polynomial fragments of SAT might possibly pave the way towards the next generation of SAT solvers, provided that the treatment of these extended fragments could be grafted within the search for satisfiability or included in some pre-processing steps.

In this respect, the focus in this thesis is on the linear time unit propagation (in short, UP) inference rule, which is recognized as being a fundamental and elementary step of all state-of-the-art satisfiability solvers. More precisely, we attempt to extend current polynomial fragments of SAT thanks to UP in such a way that the fragments can still be recognized and solved in polynomial time. In addition to its fundamental role inside SAT solvers, UP is also an important inference rule for many tractable classes such as Horn formulas. Indeed, UP is a basic operation in all DPLL-like procedures and modern SAT solvers. A linear-time algorithm for Horn satisfiability, one of the most popular tractable fragments of SAT, is also based on this inference rule and exploited in this thesis.

Specifically, our first contribution focuses on the so-called Quad fragments of SAT, which have been proposed in [6] as “almost" quadratic fragments of SAT. The Quad tractable class makes use of UP to infer sub-clauses until the formula is reduced into a member of a “root" class that is made of only binary clauses, or formulas without positive or negative clauses. In this last case, the formula is known to be satisfiable. Firstly, we establish some properties of Quad fragments. Secondly, we extend these fragments and exhibit promising variants. More precisely, we start by studying the sensitivity of Quad fragments to clause elimination and variable assignment. Then, an extension is obtained by allowing Quad fixed total orderings of clauses to be accompanied with specific additional separate orderings of maximal sub-clauses, i.e., sub-clauses obtained by removing only one literal. Interestingly, the resulting fragments extend Quad without degrading its worst-case complexity. Finally, we question other fundamental principles that are grounding Quad and that could be relaxed while keeping the polynomial time complexity. Especially, we investigate how bounded resolution and redundancy through unit propagation can play a role in this respect.

Our second contribution on tractable sub-classes of SAT is obtained by extending the well-known Tovey’s polynomial fragment [9] so that it also includes instances that can be simplified using UP. Then, we compare two existing polynomial fragments based on UP, namely, Quad [6] and UP-Horn [10]. Interestingly, many benchmarks [11] from SAT competitions [7] are shown to belong to UP-Horn. This result can be interpreted as a step towards the integration of theoretical investigations about tractable fragments within practical SAT solving. We also answer an open question about the connections between these two classes: we show that UP-Horn and some other UP-based variants are strict subclasses of $\bigcup$Quad, where $\bigcup$Quad is the union of all Quad classes obtained by investigating all possible orderings of clauses.

The dissertation thesis is organized as follows. After this general introduction, the thesis is divided into three other parts. Part II contains two chapters: the first one is dedicated to propositional logic while chapter 2 introduces the SAT problem and the main techniques used in SAT solvers. In part III, we

introduce in chapter 3 the tractable subclasses of SAT and focus on the tractable subclasses that use the UP technique. Finally in part IV we introduce our contributions. Chapter 4 introduces our extensions and variants of Dalal's Quad tractable class and in chapter 5 we present UP-based polynomial fragments of SAT whereas chapter 6 is dedicated to extensions of Tovey's polynomial fragment of SAT. In the conclusive chapter, we present some promising paths for further research.

Part II

Propositional logic and SAT

Chapter 1

Propositional logic

In this chapter, we present the main concepts useful for this thesis that are related to standard propositional (or, Boolean) logic, which is the branch of mathematical logic introduced in its modern form by G. Boole [12]. As the goal of this introductory chapter is to recall briefly the fundamentals of propositional logic, we only present the notations, conventions, definitions and properties required for reading this thesis, as so many works (see for example [13]) already cover this subject very clearly and exhaustively.

Propositional logic allows the study of assertions that are based on propositional variables, which are variables that can be assigned to one among two values only: *TRUE* and *FALSE*. The propositional variables are linked together by logical connectives: the negation: $\neg$ (*not* ...), the conjunction: $\wedge$ (...*and*...), the disjunction: $\vee$ (...*or*...), the (material) implication: $\rightarrow$ (*if* ...*then*...) and the equivalence connective: $\leftrightarrow$ (... *if and only if* ...) to construct formulas. The truth value of formulas (*TRUE* or *FALSE*) depends on the truth value of their components.

First, we present the syntax and semantic of propositional logic [14]. After that, we describe the classical simplifications of propositional formulas under conjunctive normal form that preserve satisfiability.

1.1 Syntax

Definition 1. *(Atom)*
An atom is a propositional variable, which can be assigned a truth value. We will denote the truth values by $\{FALSE, TRUE\}$ *or* $\{F, T\}$ *or* $\{0, 1\}$*, indifferently.*

Definition 2. *(Formula)*
Let α *be an alphabet with a finite set of atoms and with the operators:*

- *The negation:* $\neg$ (*not*...)
- *The conjunction:* $\wedge$ (...*and*...)
- *The disjunction:* $\vee$ (...*or*...)
- *The implication:* $\rightarrow$ (*if*...*then*...)
- *The equivalence:* $\leftrightarrow$ (...*if and only if*...)

And the auxiliary parenthesis symbols "(" and ")".
A propositional formula is recursively constructed by applying a finite number of times the following rules:

1. *Atoms* $p, q, \ldots$ *are formulas;*
2. *If* Σ *is a formula then* (Σ) *is a formula;*
3. *If* Σ *is a formula then* $\neg\Sigma$ *is a formula;*
4. *If* Σ *and* Σ' *are formulas then:*
   - $\Sigma \wedge \Sigma'$ *is a formula;*
   - $\Sigma \vee \Sigma'$ *is a formula;*
   - $\Sigma \rightarrow \Sigma'$ *is a formula;*

- $\Sigma \leftrightarrow \Sigma'$ *is a formula.*

Definition 3. *(Literal)*
A literal is an atom p or its negation $\neg p$: p is called a positive literal and $\neg p$ is called a negative literal. p and $\neg p$ are called complementary and we denote $\neg\ell$ the literal complementary to the literal ℓ. Obviously, $\neg\neg\ell$ is equal to ℓ.

Definition 4. *(Pure (or Monotone) Literal)*
A literal ℓ is said to be a pure (or a monotone) in a formula Σ if and only if ℓ appears in Σ and $\neg\ell$ does not appear in Σ.

1.2 Semantics

1.2.1 Interpretation of a Formula

Definition 5. *(Interpretation)*
Let Σ be a propositional formula and $Vars(\Sigma)$ the set of propositional variables occurring in Σ. An interpretation (or a truth assignment) of Σ is a function from the set of propositional variables $Vars(\Sigma)$ to the set of truth values $\{F, T\}$.
The interpretation $I(\Sigma)$ of a formula Σ can then be defined by the truth values of the propositional variables of Σ.
$I(\Sigma)$ is calculated according to the following rules:

1. $I(\top) = T$;
2. $I(\bot) = F$;
3. $I(\neg\Sigma) = T$ *if and only if* $I(\Sigma) = F$;
4. $I(\Sigma \wedge \Sigma') = T$ *if and only if* $I(\Sigma) = I(\Sigma') = T$;
5. $I(\Sigma \vee \Sigma') = F$ *if and only if* $I(\Sigma) = I(\Sigma') = F$;
6. $I(\Sigma \rightarrow \Sigma') = F$ *if and only if* $I(\Sigma) = T$ *and* $I(\Sigma') = F$;
7. $I(\Sigma \leftrightarrow \Sigma') = T$ *if and only if* $I(\Sigma) = I(\Sigma')$.

Notation 1. *Let Σ be a formula and I be an interpretation of Σ. I is called a model of Σ, written $I \models \Sigma$, if and only if $I(\Sigma) = T$. The set of models of Σ is denoted by $\mathcal{M}(\Sigma)$.*

Remark 1.

1. *By using the values 0 and 1 for F and T respectively, we get:*
   - $I(\neg\Sigma) = 1 - I(\Sigma)$;
   - $I(\Sigma \wedge \Sigma') = min\{I(\Sigma), I(\Sigma')\}$;
   - $I(\Sigma \vee \Sigma') = max\{I(\Sigma), I(\Sigma')\}$.
2. - $(\Sigma \rightarrow \Sigma') \equiv (\neg\Sigma \vee \Sigma')$;
   - $(\Sigma \leftrightarrow \Sigma') \equiv ((\Sigma \rightarrow \Sigma') \wedge (\Sigma' \rightarrow \Sigma)) \equiv ((\neg\Sigma \vee \Sigma') \wedge (\neg\Sigma' \vee \Sigma))$.
   - *The exclusive or (for the difference) of two formulas, denoted $(\Sigma \oplus \Sigma')$, is the formula:* $\neg(\Sigma \leftrightarrow \Sigma')$.
     And if we restrict ourselves to the use of the operators $\{\neg, \wedge, \vee\}$, then it is the formula $((\neg\Sigma \wedge \Sigma') \vee (\neg\Sigma' \wedge \Sigma))$.
3. *We denote by $M_{I(\Sigma)}$ the set of interpretations of a formula Σ. If Σ has exactly n different atoms then* $|M_{I(\Sigma)}| = 2^n$.

Remark 2. *In propositional calculus, in order to describe an interpretation it is sufficient to know the truth value given to all the propositional variables occurring in the formula. So, it is natural to characterize an interpretation I by the set of atoms assigned to T. For example, for the formula* $(a \vee b) \rightarrow (c \rightarrow d)$ *and the interpretation* $I(a) = T, I(b) = T, I(c) = F, I(d) = F$*, I is denoted by* $\{a, b\}$*. However, for incomplete interpretations (see definition 6) this representation cannot distinguish between the atoms that take the truth value FALSE and the unassigned atoms. So we adopt the convention of representing an interpretation by its true literals. Hence, the interpretation I above will be represented by the set* $\{a, b, \neg c, \neg d\}$.

Definition 6. *(Complete and Incomplete Interpretation)*
Let Σ *be a formula with n distinct atoms.*

1. *An interpretation I (represented as a set of literals) is complete if and only if it has non complementary n distinct literals.*
2. *An interpretation* I' *is incomplete if and only if there is a complete interpretation I such that* $I' \subset I$.

Remark 3. *Let* Σ *be a formula with n distinct atoms then*

- *If I is a complete interpretation then* $|I| = n$.
- *If I is an incomplete interpretation then* $|I| < n$.

1.2.2 Consistency and Inconsistency of a Formula

Definition 7. *(Satisfied formula)*
A formula Σ *is satisfied by an interpretation I if and only if* $I(\Sigma) = T$.

Definition 8. *(Falsified formula)*
A formula Σ *is falsified by an interpretation I if and only if* $I(\Sigma) = F$.

The concepts of consistency and inconsistency of a formula are defined as follows:

Definition 9. *(Consistency/Inconsistency)*

- *A formula* Σ *is called consistent (or satisfiable) if and only if* Σ *admits a model.*
- *A formula* Σ *is called inconsistent (or unsatisfiable) if and only if* Σ *admits no model, that is* $\mathcal{M}(\Sigma) = \emptyset$.

Notes 1. *(Tautological Formula)*
A formula is tautological if and only if $\mathcal{M}(\Sigma) = M_I(\Sigma)$.

1.2.3 Logical Consequence

Definition 10. *(Logical Consequence)*
A formula Σ *semantically implies the formula* Σ'*, denoted* $\Sigma \models \Sigma'$*, if and only if* $\mathcal{M}(\Sigma) \subseteq \mathcal{M}(\Sigma')$*. In this case we say that* Σ' *is a logical consequence of* Σ.

Definition 11. *(Implied Literal)*
A literal l is implied by a formula Σ*, denoted* $\Sigma \models l$*, if and only if l appears in every model of* Σ.

Definition 12. *(Logical Equivalence)*
Two formulas Σ *and* Σ' *are logically equivalent, denoted* $\Sigma \equiv \Sigma'$*, if and only if* $\Sigma \models \Sigma'$ *and* $\Sigma' \models \Sigma$*, i.e.,* $\mathcal{M}(\Sigma) = \mathcal{M}(\Sigma')$.

Example 1. *A trivial example of semantics equivalence of two formulas:* $(\Sigma \rightarrow \Sigma') \equiv (\neg\Sigma \vee \Sigma')$.

The following property can be proved by a contradiction.

Property 1. *Let* Σ *and* Σ' *be two formulas:* $\Sigma \models \Sigma'$ *if and only if* $\Sigma \wedge \neg\Sigma'$ *is inconsistent.*

1.2.4 Clauses, Cubes and Normal Forms

Clauses and Cubes

Definition 13. *(Clause)*
A clause is a finite disjunction of literals.

Definition 14. *(Cube)*
A cube is a finite conjunction of literals.

Remark 4. *We denote a clause either by a finite set of literals* $\{l_1, l_2, \ldots, l_n\}$ *or by its disjunction* $(l_1 \vee l_2, \cdots \vee l_n)$.

Definition 15. *(Fundamental Clause and Cube)*
A clause (resp. cube) is fundamental if it is a clause (resp. cube) which does not contain complementary literals.

In general, we say that a set of literals is fundamental if it is a set which does not contain complementary literals. In particular an interpretation is a fundamental set of literals.

Remark 5. *A clause which is not fundamental is a tautology, so the fundamental clause can be falsified. A cube which is not fundamental is inconsistent, so the fundamental cube can be satisfied.*

Definition 16. *(Positive, Negative and Mixed Clause)*
A clause is a positive clause if and only if all of its literals are positive and a clause is a negative clause if and only if all of its literals are negative.
A clause that contains positive and negative literals is called mixed.

Definition 17. *(Size of a Clause/Cube)*
The size of a clause C (resp. cube M) is the number of different literals in it, we denote this length by $|C|(|M|)$.

Definition 18. *(Unit Clause)*
A unit clause C is a clause of size one ($|C| = 1$). *A unit (or mono-) literal is a literal appearing in a unit clause.*

Definition 19. *(Binary Clause)*
A binary clause C is a clause of size less than or equal to 2 ($|C| \leq 2$).

Definition 20. *(Empty Clause)*
An empty clause C is a clause of size equal to zero ($|C| = 0$). *We will denote the empty clause by* $\perp$

Remark 6. *The empty clause is inconsistent (unsatisfiable).*

Definition 21. *(Empty Cube)*
An empty cube M is a cube of size equal to zero ($|M| = 0$). *We will denote the empty cube by* $\top$ *or* $\{\}$.

Notes 2. *The empty cube is always satisfiable.*

Definition 22. *(Horn and Reverse-Horn Clause)*
A clause is Horn if and only if it contains at most one positive literal.
A clause is reverse-Horn if and only if it contains at most one negative literal.

Definition 23. *(Subsumption)*
A clause C_1 *is subsumed by the clause* C_2 *if and only if* $C_2 \subseteq C_1$ *(or* C_2 *is a sub-clause of* C_1*). In this case we say that* C_2 *subsumes* C_1 *or* C_1 *is subsumed by* C_2.

Definition 24. *(Maximal Sub-Clause)*
A sub-clause C' *of C is called maximal if and only if* $|C| - |C'| = 1$.

Definition 25. *(Resolvable (clash) Clauses)*
Clauses C_1 and C_2 are resolvable (clash) if and only if there is exactly one literal l, such that $l \in C_1$ and $\neg l \in C_2$.

Definition 26. *(Resolvent of two Clauses.)*
C_1 and C_2 are resolvable where $l \in C_1$ and $\neg l \in C_2$ if and only if the resolvent of C_1 and C_2 on l is $C_1 \cup C_2 \setminus \{l, \neg l\} = (C_1 \setminus \{l\}) \cup (C_2 \setminus \{\neg l\})$ and is denoted by $\eta[l, C_1, C_2]$.

Normal Forms

Definition 27. *(CNF)*
A formula Σ is in conjunctive normal form (CNF) if and only if Σ is a conjunction of clauses, that is a conjunction of disjunctions of literals.

Definition 28. *(DNF)*
A formula Σ is in disjunctive normal form (DNF) if and only if Σ is a disjunction of cubes, that is a disjunction of conjunctions of literals.

The classical transformation of a propositional formula into CNF is exponential with respect to the length of the given formula but we can construct a CNF formula from any formula in linear time with respect to the length of the given formula by introducing new variables (see for example [15, 16]).

Notes 3. *A conjunctive (resp. disjunctive) normal form is represented by a set of clauses (resp. cubes).*

Definition 29. *(Irredundant or Minimal CNF)*
Let Σ be a CNF formula, Σ is irredundant (or minimal) if and only if for all clauses C in Σ: $(\Sigma \setminus \{C\}) \not\models C$.

Definition 30. *(Simplified CNF)*
Let Σ be a CNF and l is a literal. $\Sigma|_l$ results from Σ by removing the clauses that contain l and deleting $\neg l$ from the clauses that contain it. That is $\Sigma|_l$ is the formula obtained from Σ by assigning l the truth-value $TRUE$. Formally $\Sigma|_l = \{C \setminus \{\neg l\} \mid C \in \Sigma \text{ and } l \notin C\}$.

Example 2. *If $\Sigma = \{\{x_1, x_2\}, \{x_1, \neg x_2\}, \{\neg x_1, x_3\}, \{x_3, x_4\}\}$ then $\Sigma|_{x_1} = \{\{x_3\}, \{x_3, x_4\}\}$.*

More generally, if C is a fundamental set of literals, $\Sigma|_C$ denotes the CNF formula obtained from Σ by removing the clauses that contain l and deleting $\neg l$ from the clauses for all literals $l \in C$.

Definition 31. *(The Clauses set, the Variables set and the Literals set of a CNF)*
Let Σ be a CNF. We denote the set $\{C : C \in \Sigma\}$ by $C(\Sigma)$, the set of variables of Σ by $V(\Sigma)$ and the set of literals of Σ by $L(\Sigma)$ or L if Σ is known from the context. If $l \in L(\Sigma)$, we denote the variable in $V(\Sigma)$ associated with l by $V(l)$ and a literal in $L(\Sigma)$ associated with the variable $v \in V(\Sigma)$ by $L(v)$.
Note that
$L(v) = v$ or $L(v) = \neg v$.
$V(L(v)) = v$.
$L(V(l)) = l$ or $L(V(l)) = \neg l$.
If $l \in L(\Sigma)$ then we call l positive if $l \in V(\Sigma)$ and negative if $\neg l \in V(\Sigma)$.
If $l \in L(\Sigma)$ then $Occ_\Sigma(l)$ (or $Occ(l)$ if there is no confusion) is the set $\{C \in \Sigma | l \in C\}$ and if $v \in V(\Sigma)$, $Occ_\Sigma(v)$=$Occ_\Sigma(v) \cup Occ_\Sigma(\neg v)$, where v in the right side denotes the positive literal associated with the variable v in the left side.

1.3 The Classical Simplifications of a CNF Formula

In this section we present the classical simplifications of propositional formulas in conjunctive normal form that preserve satisfiability. They could serve as preprocessing approaches for SAT solvers (see section 2.9 for more complex preprocessing approaches).

1.3.1 Pure Literal Elimination

In a CNF Σ, we can assign the value *TRUE* to all pure literals in Σ and hence remove the clauses that contain them; this step may give rise to new pure literals. The technique of recursively applying this process until the CNF has no pure literals anymore is called pure literal elimination. The resulting CNF is implied by Σ.

1.3.2 Unit propagation (UP)

A literal in a unit clause is forced to be *TRUE*. So we can assign the value *TRUE* to all unit clauses in Σ and hence remove the clauses that contain them and remove the negation of them (which is forced to be *FALSE*) from the clauses that contain these negations; this step may result in new unit clauses. The technique of recursively applying this step until the CNF has no unit clauses anymore is called unit propagation. The resulting CNF is implied by the initial CNF.

Notation 2. *1. Σ^* denotes the CNF obtained from Σ by unit propagation and $\Sigma^\diamond$ denotes the CNF obtained from Σ by pure literal elimination, while Σ^+ denotes the CNF that results from Σ by unit propagation and pure literal elimination.*

2. *Let Σ be a CNF and Σ' result from Σ by UP, we denote that by $\Sigma \vDash^* \Sigma'$.*

3. *Without explicitly referring to the CNF under consideration, processing unit propagation on a literal a is denoted by $UP(a)$, and the notation $UP(a) \vDash l$ is used to denote that $UP(a)$ implies that the literal l becomes TRUE.*

Notes 4. *$\Sigma \vDash^* \Sigma'$ if and only if $\Sigma \wedge \neg\Sigma' \vDash^* \bot$.*

1.3.3 Adding resolvents

Notes 5. *1. If two clauses C_1 and C_2 are satisfied by a truth assignment then their resolvent is also satisfied by this truth assignment.*

2. *If C_1 and C_2 are clauses such that there exists at least two literal u_1 and u_2 such that $u_1 \in C_1$,$u_2 \in C_1$ and $\neg u_1 \in C_2$, $\neg u_2 \in C_2$ then $\eta[u_1, C_1, C_2] = \eta[u_2, C_1, C_2] = \{\}$.*

3. *Adding the resolvents preserves the satisfiability of the CNF.*

4. *If two clauses of length at most two are resolvable then their resolvent is also of length at most two.*

5. *An empty clause is obtained as a resolvent in a CNF Σ if and only if Σ is unsatisfiable.*

6. *Adding resolvents for every pair of literals l and $\neg l$ may increase the cardinality of the resulting CNF exponentially by adding a lot of clauses.*

1.3.4 Subsuming clauses

Definition 32. *(Subsumption).*
Let C_1 and C_2 be two clauses within a CNF Σ such that $C_2 \subseteq C_1$. The process of removing C_1 from Σ is called subsumption and C_2 is said to subsume C_1.

Note that if clauses C_1 and C_2 satisfy $C_2 \subseteq C_1$ then any truth assignment that satisfies C_2 satisfies C_1, too.

Remark 7. *We can combine adding resolvents as well as subsumption to simplify the given CNF instance.*

Example 3. *Let $C_1 = \{l_1, l_2, l_3, l_4\}$ and $C_2 = \{\neg l_1, l_2, l_4\}$ be two clauses in a CNF, C_1 and C_2 cannot subsume each other, but by adding the resolvent $\eta[l_1, C_1, C_2] = \{l_2, l_3, l_4\}$, the clause $C_1 = \{l_1, l_2, l_3, l_4\}$ can be subsumed by the resolvent clause $\eta[l_1, C_1, C_2] = \{l_2, l_3, l_4\}$.*

1.3.5 Binary Equivalent Literals Propagation

The binary equivalent literal propagation technique is applied for CNF formulas that contain two clauses of the form $C_1 = \{\neg l_1, l_2\}$ and $C_2 = \{l_1, \neg l_2\}$.

Definition 33. *(Equivalent literals).*
Let Σ be a CNF that contains the clauses $C_1 = \{\neg l_1, l_2\}$ and $C_2 = \{l_1, \neg l_2\}$, the literals l_1 and l_2 are called equivalent.

Note that the equivalent literals are either both *TRUE* or both *FALSE*.

Remark 8.
1. *The binary equivalent literal propagation consisting in replacing all occurrences of l_1 and $\neg l_1$ by l_2 and $\neg l_2$ respectively, and then in simplifying the given CNF according to this replacement (see section 2.9.1 for more details).*
2. *Let Σ be a CNF and $C_1 = \{\neg l_1, l_2\}$, $C_2 = \{l_1, \neg l_2\}$, $C_3 = \{\neg l_1, l_3\}$, $C_4 = \{l_2, \neg l_3\} \in \Sigma$, the first replacement changes $\{\neg l_1, l_3\}$ to $\{\neg l_2, l_3\}$. So l_2,l_3 will be equivalent literals. Hence binary equivalent literal propagation can be used to simplify the given CNF(see section 2.9.1 for more details).*

Remark 9. *We can use pure literal elimination, unit propagation, adding resolvents, subsuming clauses and binary equivalent literal propagation as preprocessing techniques.*

Chapter 2
SAT Problem

This chapter presents the SAT problem and overviews some of the recent SAT solving techniques. We begin by giving a short introduction to worst-case complexity theory for understanding the characterization of the SAT problem, then we give the definition of SAT and provide the main techniques used to solve SAT instances, such as resolution, refutation, DPLL algorithm, CDCL, branching rules and preprocessing approaches.

2.1 Complexity Classes

We present in this section some definitions and notations from complexity theory to introduce the SAT problem. For more details see [17, 18, 19, 20, 21, 22].

Definition 34. *(Turing Machine) [19, 22]*
A (one-tape deterministic) Turing machine (TM) is a 5-tuple $T = (Q, \Sigma, \Gamma, q_0, \delta)$*, where* Q *is a finite set of states.* Σ *(the input alphabet) and* Γ *(the tape alphabet) are both finite sets, with* $\Sigma \subseteq \Gamma$*.* $q_0 \in Q$ *is the initial state.*
There are two states $h_a \notin Q$ *and* $h_r \notin Q$ *(called halting states), and a blank symbol* $\Delta \notin \Gamma$*.*
δ *is the transition function:*
$\delta : Q \times (\Gamma \cup \{\Delta\}) \rightarrow (Q \cup \{h_a, h_r\}) \times (\Gamma \cup \{\Delta\}) \times \{R, L, S\}$ *where L, R and S denote the left shift, right shift and no shift, respectively.*

Definition 35. *(Nondeterministic Turing Machine)[19]*
A nondeterministic Turing machine is a Turing machine with a transition function of the form
$\delta : Q \times (\Gamma \cup \{\Delta\}) \rightarrow P((Q \cup \{h_a, h_r\}) \times (\Gamma \cup \{\Delta\}) \times \{R, L, S\})$
i.e., the transition function may take instead of one element of $Q \times (\Gamma \cup \{\Delta\})$ *a subset of this set (this set may be* $\emptyset$*, i.e., there exists a state* q *and a symbol* a *such that* $\delta(q, a) = \emptyset$*).*

Notation 3. *Let* $|x|$ *denote the length of an input* x *and* N *denote the set of natural numbers.*

Definition 36. *(The Time Complexity of a Deterministic Turing Machine)[19]*
Let M *be a deterministic Turing machine with input alphabet* Σ *that halts on every input. The time complexity of* M *is the function* $\tau_M : N \rightarrow N$ *such that if* $x \in \Sigma^*$ *is an input,* $|x| = n$ *then* $\tau_M(n)$ *is the maximum number of moves of* M *on* x *before halting.*

Definition 37. *(The Time Complexity of a Nondeterministic Turing Machine)[19, 22]*
Let M *be a nondeterministic Turing machine with input alphabet* Σ *that halts for every possible sequence of moves on every input. The time complexity of* M *is the function* $\tau_M : N \rightarrow N$ *such that if* $x \in \Sigma^*$ *is an input,* $|x| = n$ *then* $\tau_M(n)$ *is the maximum number of moves of* M *on* x *before halting, in other words if we let* τ_x *be the length of the longest sequence of moves on* x *then* $\tau_M(n) = max\{\tau_x : |x| = n\}$*.*

Definition 38. *(Big-O Notation) [17]*
Let N *and* R^+ *be the natural and positive real numbers respectively, let* f *and* g *be two functions such that* $f, g : N \rightarrow R^+$*.* $f(n) \in O(g(n))$ *if there exists positive integers* c_0 *and* n_0 *such that* $\forall n \geq n_0$*,* $f(n) \leq c_0 g(n)$*.*

Definition 39. *(The Complexity Class P) [19]*
The class P is the set of languages L such that there is a deterministic Turing machine M that decides L in time $\tau_M(n) = O(n^k)$ *for some positive integer k.*

Definition 40. *(The Complexity Class NP)[19]*
The class NP is the set of languages L such that there is a nondeterministic Turing machine M that decides L in time $\tau_M(n) = O(n^k)$ *for some positive integer k.*

Definition 41. *(Polynomial-Time Reductions)[19]*
Let L_1 *and* L_2 *be two languages over alphabets* Σ_1 *and* Σ_2 *respectively. A polynomial-time reduction from* L_1 *to* L_2 *is a function* $f : \Sigma_1^* \to \Sigma_2^*$ *such that*

1. $\forall x \in \Sigma_1^*$, $x \in L_1$ *if and only if* $f(x) \in L_2$
2. *There is a Turing machine that computes* f *in a polynomial time.*
   If there is a polynomial-time reduction from L_1 *to* L_2*, we denote that by* $L_1 \leqslant_p L_2$.

Definition 42. *(NP-Hard Languages)[19]*
A language L' *is NP-hard if and only if* $L \leqslant_p L', \forall L \in NP$.

Definition 43. *(NP-Complete Languages)[19]*
L is NP-complete if and only if $L \in NP$ *and L is NP-hard.*

Definition 44. *(SPACE)*
A problem is in the complexity class $SPACE(f(n))$ *if and only if there exists a Turing machine that solves this problem in* $O(f(n))$ *space.*

Property 2. *It is easy to check that* $P \subseteq NP$.

Definition 45. *(CoNP)*
The complexity class CoNP is the set of problems which is the complementary of problems in the class NP that is $A \in CoNP$ *if and only if* $\overline{A} \in NP$ *where* $\overline{A} = \{x : x \notin A\}$.

Property 3. *We have also* $P \subseteq CoNP$.

Corollary 1. *So,* $P \subseteq (NP \cap CoNP)$*, see figure 2.1.*

The following two conjectures follow from theoretical and practical accumulated experimental knowledge.

Conjecture 1. $P \neq NP$*, see figure 2.1.*

Conjecture 2. $NP \neq CoNP$*, see figure 2.1.*

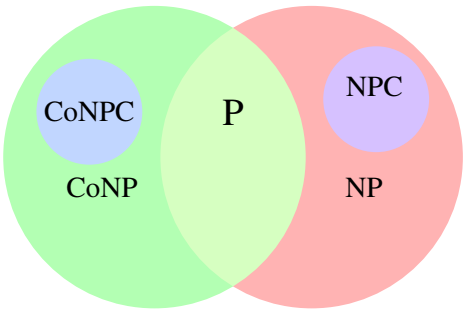

Figure 2.1: The relationship between P, NP, CoNP, NP-complete, CoNP-complete under conjecture 1 and conjecture 2

2.2 SAT: Definition

SAT is the abbreviation for SATisfiability problem of propositional formulas in conjunctive normal form. It is perhaps one of the most studied NP-complete problems and has been addressed from both theoretical and practical perspectives. Actually, it is the canonical NP-complete problems [8]. The satisfiability problem is a very important one and exhibits many applications in a lot of fields of computer science, like artificial intelligence, formal verification, hardware design, circuit design, model checking, automated planning and scheduling and automatic theorem proving. SAT was the first problem that was proved to be NP-complete. This means that there is no efficient algorithm for solving SAT unless P=NP. Most experts believe that there is no such efficient algorithm but there is no proof for this belief. In spite of this pessimistic conjecture, modern efficient SAT solvers allow us to solve SAT application instances involving millions of variables and clauses.

The definitions for SAT and kSAT (which is a special case of SAT where every clause has a length that equals to k) are as follows.

Definition 46. *(SAT)*
Instance: S is a finite set of propositional symbols and Σ is a finite set of clauses constructed from S.
Question: Does it exist an interpretation on S that satisfies the set of clauses of Σ?

Definition 47. *(kSAT)*
Instance: S is a finite set of propositional symbols and Σ is a finite set of clauses, where each clause has exactly k literals, constructed from S.
Question: Does it exist an interpretation on S that satisfies the set of clauses of Σ?

Notes 6. *SAT and kSAT are NP-complete problems [8].*

Below, we review the techniques and algorithms used for solving SAT instances.

2.3 Resolution refutation

The operation of obtaining the resolvent from two clauses in a given CNF is called the application of the resolution rule. There is an algorithm for solving SAT instances based on the resolution rule: this algorithm is called resolution refutation [23]. It consists in adding non tautological resolvents of each pair of resolvable clauses until we get the empty clause. In this last case the given CNF is unsatisfiable. Consequently, resolution is complete for refutation. In addition, if we consider also the subsumption rule, the CNF formula is answered satisfiable when no new resolvent can be derived. In the next section we present the well-known DP procedure based on the application of resolution rule to eliminate variables (see the next section 2.4). Both the time and space complexities of the resolution refutation algorithm are exponential with respect to the length of the CNF given as input.

2.4 The DP algorithm

The DP algorithm 1 introduced in [24] is based essentially on the following property.

Theorem 1. *[24]*
Let Σ be a CNF and l be a literal in Σ, rewrite Σ as following $(\Sigma_1 \vee l) \wedge (\Sigma_2 \vee \neg l) \wedge \Sigma_3$ where l does not appear (positively or negatively) in the CNFs $\Sigma_1, \Sigma_2, \Sigma_3$ then
$\Sigma = (\Sigma_1 \vee l) \wedge (\Sigma_2 \vee \neg l) \wedge \Sigma_3$ is satisfiable if and only if $(\Sigma_1 \vee \Sigma_2) \wedge \Sigma_3$ is satisfiable.

This property leads to the following procedure: Assume that Γ is a CNF that results from Σ as follows: $\forall C, D \in \Sigma$ such that C, D are resolvable clauses with the literal l, add all resolvents $\eta[l, C, D]$ to Σ and remove all such resolvable clauses.
Γ is satisfiable if and only if Σ is satisfiable.
We can implement DP by using the so-called bucket elimination procedure [25].

Procedure 1. *(Bucket Elimination Procedure) [25].*

1. *Order the variables using a fixed total order* Π.
2. *For each variable v in the CNF, construct a bucket and label it with v.*
3. *Sort the buckets from top to bottom according to their labels using the order* Π.
4. *Add the clause in the CNF that contains the variable v to the first bucket v from the top.*

Algorithm 1: DP algorithm $DP(\Sigma)$[25]

```
1  Input: A CNF formula Σ, a variable ordering Π;
2  Ouput: The CNF formula Σ is SAT or UNSAT;
3  for each variable v of Σ do
4      create empty bucket B_v;
5  for each clause C of Σ do
6      v← first variable of C according to order Π;
7      B_v ← B_v ∪ {C};
8  for each variable v in Σ according to the order Π do
9      if B_v is not empty then
10         for each v-resolvent C of clauses in B_v do
11             if C is the empty clause then
12                 return UNSAT;
13             u← first variable of C according to order Π;
14             B_u ← B_u ∪ {C};
15 return SAT ;
```

Remark 10. *In the DP procedure, the ordering* Π *can be set dynamically. The efficiency of such procedure heavily depends on such static or dynamic ordering.*

2.5 The DPLL algorithm

The Davis-Putnam-Logemann-Loveland (DPLL) algorithm was introduced by Martin Davis, Hilary Putnam, George Logemann and Donald W. Loveland in [26]. This procedure can be seen as a variant of the Davis-Putnam's one [24]. The DPLL algorithm is at the basis of most modern SAT solvers such as Chaff [27], GRASP [28], Satz [29], Minisat [30, 31, 32, 33]. The DPLL algorithm is a complete backtracking search algorithm that decides SAT instances. It starts by choosing a literal, assigns a truth value to it, simplifies the CNF by

1. removing the clauses that become *TRUE* under the assignment of the chosen literal.
2. removing the literals that become *FALSE* from the remaining clauses

Then this algorithm checks if the simplified formula is satisfiable:

1. if all the clauses are satisfied then the input CNF is also satisfiable.
2. otherwise, it chooses the other (opposite) truth value of the literal under consideration and applies the procedure recursively.

DPLL algorithm is introduced in algorithm 2. The cornerstone of DPLL algorithm is the heuristic of branching rules. Naturally, the DPLL procedure includes the well-known unit propagation rule (lines 3-7). Pure literals can also be propagated at each step. DPLL was a very popular complete method for SAT in the last century. Now the most efficient complete method for SAT is based on CDCL (Conflict Driven Clause Learning), see section 2.6.

Algorithm 2: DPLL Algorithm $DPLL(\Sigma)$

```
Input: A CNF formula Σ;
Ouput: The CNF formula Σ is SAT or UNSAT;
while Σ includes a clause C such that |C| ≤ 1 do
    if C = ∅ then
        return UNSAT ;
    if C = {x} then
        Σ ← Σ|_x ;
if Σ = ∅ then
    return SAT ;
Choose a literal u depending on the condition of the given branching rule ;
if DPLL(Σ|_u) = SAT then
    return SAT ;
if DPLL(Σ|_ū) = SAT then
    return SAT ;
return UNSAT ;
```

2.6 Conflict Driven Clause Learning (CDCL)

By combining the DPLL and resolution refutation algorithms, we obtain a special case of DPLL algorithm, called Conflict Driven Clause Learning (CDCL). In this section we present CDCL; most definitions, notions and notations are taken from [34]. If a clause with all its literals are *FALSE* (a conflicting clause) is obtained by DPLL then new clauses (learned clauses [conflict clauses]) are added to the given CNF.

Definition 48. *(Decision Levels of Truth Assignments)*

1. *(level 0) The truth assignments made before the first decision are set to level 0;*
2. *(level n) By induction, the truth assignments (decision/unit propagation) performed at the nth recursive call to DPLL are set to level n.*

Definition 49. *(Antecedent [or Reason Clause] of a Truth Assignment)*
The clause that became a unit clause and leads to unit propagation of the remaining unit literal is called the antecedent clause of this literal.

Notes 7. *A decision literal has no antecedent clause.*

Definition 50. *(Partial Ordered Interpretation) [34]*
At level i, the current partial assignment ρ is represented as a sequence of decision-propagation of the form $\langle (x_k^i), x_{k_1}^i, x_{k_2}^i, \ldots, x_{k_{n_k}}^i \rangle$, where the first literal x_k^i corresponds to the decision literal x_k assigned at level k and each $x_{k_j}^i$ for $1 \leq j \leq n_k$ represents propagated (unit) literals at level k, is called partial ordered interpretation.

Notation 4.
1. *Let ρ be an interpretation and $x \in \rho$. $l(x)$ denote the assignment level of x.*
2. *Assume that the literal y is propagated, $\overrightarrow{imp}(y)$ denotes the clause of the form $(x_1 \vee \cdots \vee x_n \vee y)$ such that every literal x_i is FALSE under the current partial interpretation and $\rho(y) = TRUE$*
3. *we use the notation $\overrightarrow{imp}(y) = \perp$ if the literal y is not obtained from propagation but from a decision (note that in this case $\overrightarrow{imp}(y)$ is undefined).*

4. $d(\rho, i) = x$ *if* x *is the decision literal assigned at level* i.

Definition 51. (Explanations)*[34]*
Let $\overrightarrow{imp}(y) \neq \bot$, *the* explanations *is the set* $\{\overline{x} \mid x \in \overrightarrow{imp}(y) \setminus \{y\}\}$ *denoted by* $exp(y)$.

Definition 52. *(Implication Graph) [34]*
Let Σ *be a CNF formula,* ρ *a partial ordered interpretation, and let* exp *denote the set of explanations for the unit propagated literals in* ρ. *The implication graph associated to* Σ, ρ *and* exp *is* $\mathcal{G}_\Sigma^{(\rho,exp)} = (\mathcal{N}, \mathcal{E})$ *where:*

- $\mathcal{N} = \rho$, *i.e., there is exactly one node for every literal, decision or implied;*
- $\mathcal{E} = \{(x, y) \mid x \in \rho, y \in \rho, x \in exp(y)\}$

The graph $\mathcal{G}_\Sigma^{(\rho,exp)}$ will denote by $\mathcal{G}_\Sigma^{\rho}$.

Definition 53. *(Conflict Graph)*
A conflict graph is an implication graph that contains a conflit (a unit literal and its negation).

Definition 54. *(Conflict Side)*
A subgraph of a conflict graph that contains conflicting assignments is called a conflict side.

Definition 55. *(Reason Side)*
A subgraph of a conflict graph that contains decision assignments is called a reason side.

Definition 56. *(Cut)*
A partition of the conflict graph into a conflict side and a reason side is called a cut.

Definition 57. *(Vertex of cut)*
A vertex in a reason side that has at least one edge in conflict side is called a vertex of cut.

Notes 8. *(Construction of a Learned Clause (a Conflict Clause)).*
The learned clause consists of the negations of the vertices of a cut.

Notes 9.
1. *Different cuts leads to different learned clauses.*
2. *We can add the learned clauses to the given CNF and preserve the satisfiability and the number of the models of the given CNF.*
   Adding the learned clauses is very helpful technique in the modern SAT solvers.

A learned clause that has only one literal from the current decision level is called asserting clause.

Definition 58. *(Asserting Clause) [34]*
A conflict clause c *of the form* $(\alpha \vee x)$ *is called an asserting clause if and only if* $\rho(c) = FALSE$, $l(x) = m$ *and* $\forall y \in \alpha, l(y) < l(x)$. x *is called asserting literal.*

Definition 59. *(Asserting Clause Derivation)[34]*
An asserting clause derivation $\pi(\sigma_k)$ *is a sequence of clauses* $\langle \sigma_1, \sigma_2, \ldots \sigma_k \rangle$ *satisfying the following conditions :*

1. $\sigma_1 = \eta[x, \overrightarrow{imp}(x), \overrightarrow{imp}(\neg x)]$, *where* $\{x, \neg x\}$ *is the conflict.*
2. σ_i, *for* $i \in 2..k$, *is built by selecting a literal* $y \in \sigma_{i-1}$ *for which* $\overrightarrow{imp}(\overline{y})$ *is defined. We then have* $y \in \sigma_{i-1}$ *and* $\overline{y} \in \overrightarrow{imp}(\overline{y})$: *the two clauses resolve. The clause* σ_i *is defined as* $\eta[y, \sigma_{i-1}, \overrightarrow{imp}(\overline{y})]$;
3. σ_k *is an asserting clause.*

Definition 60. *(Elementary asserting clause derivation)*
An asserting clause derivation $\pi(\sigma_k) = \langle \sigma_1, \sigma_2, \ldots \sigma_k \rangle$ *is called elementary if and only if* $\exists i < k$ *s.t.* $\pi(\sigma_i) \subset \pi(\sigma_k)$ *is also an asserting clause derivation.*

A unique implication point (UIP) is a vertex in a conflict graph such that the directed paths from the decision vertex to conflict vertices pass through this vertex.

Definition 61. *(Unique Implication Point [UIP]) [34]*
A node $x \in \mathcal{N}$ is a UIP if and only if there exists an asserting clause derivation $\langle \sigma_1, \sigma_2, \ldots, \sigma_k \rangle$ s.t. $\overline{x} \in \sigma_k$ and $l(x)$ is equal to the current decision level, m. (Note that σ_k, being assertive, has exactly one such x.)

Notes 10. 1. *We can use UIP to construct a cut.*

2. *There are different methods to get a cut from UIP, for example*

   - *A cut is constructed at the closest UIP to the conflict (the so-called the first UIP).*
   - *A cut is constructed such that the learned clause is asserting and its length is minimal.*

Definition 62. *(First Unique Implication Point) [34]*
A node $x \in \mathcal{N}$ is a First UIP if and only if it is obtained from an elementary *asserting clause derivation; i.e. $\exists \pi(\sigma_k) = \langle \sigma_1, \sigma_2, \ldots, \sigma_k \rangle$ an elementary asserting clause derivation s.t. $\overline{x} \in \sigma_k$ and $l(x) = m$.*

Definition 63. *(Last Unique Implication Point) [34]*
The last UIP is defined as the literal $d(\rho, m)$, i.e., the decision literal assigned at the conflict level m.

Example 4. *Let Σ be a CNF and ρ be the partial assignment $\rho = \{\langle \neg x_1^1 \ldots \rangle \langle (x_2^2) \ldots x_3^2 \ldots \rangle \langle (x_4^3) \ldots \rangle \ldots \langle (x_5^5) \ldots \rangle\}$ where the current decision level is 5. Let $C_1, \ldots, C_{10}$ be clauses in Σ where*

$$
\begin{array}{llll}
(C_1) & \neg x_2 \vee \neg x_5 \vee x_6 & (C_2) & \neg x_5 \vee x_7 \\
(C_3) & \neg x_6 \vee \neg x_7 \vee x_8 & (C_4) & \neg x_3 \vee \neg x_8 \vee x_9 \\
(C_5) & \neg x_1 \vee \neg x_8 \vee x_{10} & (C_6) & \neg x_4 \vee \neg x_9 \vee x_{11} \\
(C_7) & \neg x_4 \vee \neg x_9 \vee \neg x_{10} \vee x_{12} & (C_8) & \neg x_{10} \vee x_{13} \\
(C_9) & \neg x_{11} \vee \neg x_{12} \vee x_{14} & (C_{10}) & \neg x_{13} \vee \neg x_{14}
\end{array}
$$

The graph $\mathcal{G}_\Sigma^\rho$ in figure 2.2 is an implication graph.

$(\neg x_1^1 \vee \neg x_3^2 \vee \neg x_4^3 \vee \neg x_8^5)$ is a first asserting clause, because all its literals are assigned before the current level except (x_8^5) which is assigned a the current level 5.

x_8^5 is a first UIP.

$(\neg x_1^1 \vee \neg x_2^2 \vee \neg x_3^2 \vee \neg x_4^3 \vee \neg x_5^5)$ is asserting clause that corresponds to the UIP x_5^5.

The UIP x_5^5 is the last UIP because it corresponds to the decision literal.

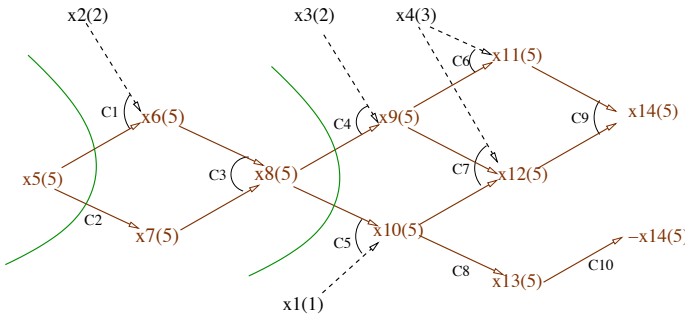

Figure 2.2: Implication Graph $\mathcal{G}_\Sigma^\rho = (\mathcal{N}, \mathcal{E})$

2.7 Modern SAT solvers

In this section we introduce the components of modern SAT solvers (see books [35, 25] for more details). We review briefly the following components, see [34] for more details .

- Restart policies
- Lazy Data Structures - Watched Literals

For other components as branching rules and preprocessing approaches, see section 2.8 and section 2.9 respectively.

2.7.1 Restart strategies

Gomes et al. in [36] have shown that different variable orderings may lead to a dramatic change in the time needed to solve SAT instances from many domains: from a polynomial one to an exponential one. This observation was at the origine of the integration of restart strategies in modern SAT solvers. The restart strategies in modern SAT solvers have the following key aspects [34].

- The search starts at the root of the search tree.
- The informations gathered from the previous search steps (like learned clauses and variables activities) are maintained.
- Restart the search on the most actives variables.
- Compact the assignment stack.
- Improve the order of assumptions.

2.7.2 Lazy Data Structures in modern SAT solvers

The performance of an implementation of the CDCL algorithm depends also on the data structures. For a survey see [37]. In this section some of the data structures integrated in SAT solvers are described [37, 38].

Adjacency Lists

Adjacency lists is a data structure where clauses are represented as lists of literals and each variable x contains a complete list of clauses that contain the literal x or the literal $\neg x$.

Assigned Literal Hiding

An assigned literal hiding maintains for each clause three lists: unassigned, assigned $TRUE$ and assigned $FALSE$.
A clause is satisfied if at least one of its literals are assigned $TRUE$, unsatisfied if all its literals are assigned $FALSE$, and unit if exactly one literal is unassigned and the remaining are assigned $FALSE$.

The Counter-based Approach

The counter-based approach is another way to get unsatisfied, satisfied and unit clauses. Support each clause with two counters (n_t, n_f) where n_t is the number of literals assigned TRUE and n_f is the number of literals assigned FALSE.
The clause C is unsatisfied if $n_f = |C|$, satisfied if $n_t \geq 1$, and unit if $n_f = |C| - 1$ and $n_t = 0$ [28].
When a clause becomes unit, it has to check which literal needs to be set to $TRUE$ [38].

Counter-based Approach with Satisfied Clause Hiding and Assigned Literal Hiding

If a clause C becomes satisfied, it will hidden from the list of clauses of all the literals in it.This technique is used [39] to avoid the problem that a clause is satisfied by more than one of its literals. Also in this technique literals that are assigned $FALSE$ are hidden from the list of literals in clauses.

Lazy data structures

A literal x may have a reference to a large number of clauses. This is a problem for adjacency list-based data structures, since most of these clauses need not be analyzed when x is assigned. Lazy data structures is presented to solve this problem such that a literal maintains a reduced set of clause references. Examples of lazy data structures are Head-Tail lists data structure [40] and two watched literals method [41].

Two Watched Literals

The idea is the following [34]:

- Watch two unassigned literals for each active clause.
- The clause that contains the two watched literals cannot be used in unit propagation.
- If backtracking is done then no update is needed.

The use of the two watched literals method often achieves significant reductions of the computation time.

2.8 Branching Rules

Many heuristics are proposed to serve as branching rules in SAT solvers (that is to choose the next literal to assign). We review in this section some of the well-known branching rules (see [42, 43, 44, 45, 46] for more details). Branching on a literal x means calling $DPLL(\Sigma|x)$, and if we get a contradiction, we call $DPLL(\Sigma|\neg x)$).

2.8.1 A branching Rules Model

Ouyang in [47] introduced a method for choosing a literal that serves as a paradigm for branching rules. This paradigm consists of :

1. A literal-weight $w(\Sigma, x)$, where Σ is a CNF and x is a literal.
2. A function Φ that counts the score of given variable (aggregates the weight of the two associated literals).

The paradigm works by :

1. Finding a variable x that maximizes $\Phi(w(\Sigma, x), w(\Sigma, \overline{x}))$.
2. Choosing x if $w(\Sigma, x) \geq w(\Sigma, \overline{x})$ and choosing $\overline{x}$ otherwise.

If more than one variable maximizes Φ then a variable is selected randomly or a new heuristic, called tie-breaker, is used.

2.8.2 VSIDS Rule

The VSIDS branching rule is a popular branching rule introduced in chaff [27] and it most widely used in the modern SAT solvers. It can defined as follows.

Definition 64. *(VSIDS Branching Rule) [27]*
VSIDS branching rule consists of the following steps.

1. *introduce an activity for each literal and initialize it to 0.*
2. *When a learnt clause is generated, increment the activity of the literals appearing in it.*
3. *Choose an unassigned literal that has the highest activity.*

2.9 Preprocessing approaches

Before calling DPLL or CDCL some preprocessing techniques are used to simplify the formula while preserving satisfiability. In this section, we describe some of these preprocessing techniques (see for example [48, 49, 38, 50, 51, 52, 53, 54, 55, 56] for more details).

2.9.1 HypBinRes+Eq: Hyper-Resolution and Equality Reduction

In this section, we describe HypBinRes as introduced in [50] which is a particular hyper-resolution rule, augmented with equality reduction for preprocessing the input CNF.
HypBinRes is the process for implying new binary clauses and then to use these binary clauses to determine,

1. implied literals, or
2. equivalent literals.

In each of these two cases, the given CNF can then be simplified to one that does not contain these kinds of literals [50]; for more details see [50, 51, 52]. In [50] three techniques, HypBinRes, equality reduction and unit clauses reduction are used. They used these three rules to build a HyperPreprocessing Algorithm which is an efficient algorithm for simplifying the initial CNF.

HypBinRes

Definition 65. *(Resolution)[50].*
The resolution is the inference rule of producing the resolvent of two clash clauses, see section 1.3.3.

Definition 66. *(Hyper-Resolution).*
The hyper-resolution inference rule is a resolution rule that involves more than two clauses

Definition 67. *(HypBinRes)[50].*
HypBinRes inference rule is a hyper-resolution inference rule that takes as input a single n-ary clause $(n \geq 2)$ $\{l_1, l_2, \ldots, l_n\}$ *and* $n-1$ *binary clauses* $\{\neg l_i, \ell\}$,$(i = 1, \ldots, n-1)$ *and output the binary clause* $\{\ell, l_n\}$.

Example 5. *Let* $\{a, b, c, d\}, \{\neg a, e\}, \{\neg b, e\}, \{\neg c, e\}$ *be the input clauses, then HypBinRes produces the binary clause* $\{d, e\}$.

Remark 11. *[50].*

1. *If we consider one more binary clause then HypBinRes can be used to obtain unit clauses, for example with* $\{a, b, c, d\}, \{\neg a, e\}, \{\neg b, e\}, \{\neg c, e\}, \{\neg d, e\}$, *hyper-resolution yields the unit clause* $\{e\}$. *Equivalently,*

   (a) *First by using HypBinRes on* $\{a, b, c, d\}, \{\neg a, e\}, \{\neg b, e\}, \{\neg c, e\}$ *we get the binary clause* $\{d, e\}$.

   (b) *Second by using an ordinary resolution step with this binary clause* $\{d, e\}$ *with the clause* $\{\neg d, e\}$ *yields the unit clause* $\{e\}$.

2. *If the input n-ary clause is binary, HypBinRes reduces to the resolution of binary clauses. for example if the n-ary clause is* $\{a, b\}$ *then with the clause* $\{\neg a, e\}$, *HypBinRes produces the binary clause* $\{a, e\}$.

Equality Reduction

Definition 68. *(Equality Reduction) [50].*
Equality reduction is the binary equivalent literal propagation (see section 1.3.5), so if $\{\neg a, b\}$ and $\{a, \neg b\}$ are clauses of CNF Σ, Equality reduction of a CNF Σ (denoted by EqReduce(Σ) [50]

1. *First replace all occurrences of b in Σ by a.*
2. *Then remove the clauses that contain both a and $\neg a$.*
3. *Third remove the duplicate of a (or $\neg a$) from the clauses.*

Example 6. *Let $\Sigma = \{\{a, \neg b\}, \{\neg a, b\}, \{a, \neg b, c\}, \{b, \neg d\}, \{a, b, d\}\}$ then $EqReduce(\Sigma) = \{\{a, \neg d\}, \{a, d\}\}$*

Unit Clauses Reduction

Definition 69. *[50]*
Unit clauses reduction is a one step of unit propagation (see section 1.3.2). So if a CNF contains a unit clause $\{l\}$ then unit clauses reduction (denoted by $UR(l)$ [50]) consists in removing all clauses containing l, and then in removing $\neg l$ from all remaining clauses.

HypBinRes+Eq Closure

Definition 70. *(HypBinRes+Eq Closure) [50]*
HypBinRes+eq closure is the result of applying unit reduction, HypBinRes, and equality reduction rules to a CNF until no more new inferences can be made with these rules.

Definition 71. *(HypBinRes+Eq Closed CNF) [50]*
A CNF is called HypBinRes+eq closed if and only if applying unit reduction, HypBinRes, and equality reduction to it infers nothing new.

Theorem 2. *[50]*
The HypBinRes+eq closure of a CNF Σ is unique up to renaming. That is, the order in which the inference rules are applied is irrelevant, as long as we continue until we cannot apply them anymore.

Theorem 2 allows these inference rules to be freely applied in any order.

UP and HypBinRes+Eq

In this section the relationship between UP and HypBinRes is presented [50]. The authors in [50] gave the following theorem that explains the relationship between UP and HypBinRes.

Theorem 3. *UP is more powerful than a single HypBinRes resolution step, but not as powerful as a sequence of HypBinRes resolution steps.*

More precisely:

1. If $\{a, b\}$ can be produced by a single HypBinRes step, then either $UP(\neg a) \vDash b$ or $UP(\neg b) \vDash a$.
2. There are CNFs from which a binary clause $\{a, b\}$ can be produced from a sequence of HypBinRes steps, but neither $UP(\neg a) \vDash b$ nor $UP(\neg b) \vDash a$.
3. In a CNF with no unit clauses (we can remove all units by unit propagation phase), if $UP(\neg a) \vDash b$ then there is a sequence of HypBinRes steps that produce $\{a, b\}$.

Performing HypBinRes+Eq Closure with UP

We can obtain HypBinRes+Eq closure by repeatedly applying HypBinRes, unit resolution, and equality reduction until we could not conclude anything more from these three rules.
By Theorem 3 we can get HypBinRes closure by repeatedly applying UP on the literals of the CNF, see [50] for more details.

2.9.2 NiVER: Non Increasing Variable Elimination Resolution

The preprocessing NiVER focuses on variables elimination [49]. On application SAT instances, it confers reducing the number of variables up 58%, the number of clauses up to 46% and the literal count by 74% [49].
See algorithm 3

Algorithm 3: NiVER CNF preprocessor[49]

```
1  Input: a CNF formula Σ;
2  Output: a simplified CNF formula;
3  repeat
4      entry ← FALSE;
5      forall the v ∈ Var(Σ) do
6          P_C ← {C : C ∈ Σ, l_v ∈ C};
7          N_C ← {C : C ∈ Σ, ¬l_v ∈ C} ;
8          R ← {};
9          forall the P ∈ P_C do
10             forall the N ∈ N_C do
11                 R ← R ∪ Resolve(P, N);
12                 Old − Num − Lits ← Number of literals in (P_C ∪ N_C);
13                 New − Num − Lits ← Number of literals in R;
14                 if Old − Num − Lits ≥ New − Num − Lits then
15                     Σ ← Σ − (P_C ∪ N_C), Σ ← Σ ∪ R, entry ← TRUE;
16 until ¬ entry;
17 return Σ;
```

Notes 11. *([49])*
Some notes are necessary to explain NiVER and how to deal with variables in the CNF.

1. *First it does not focus on the occurrences of variables (it sometimes removes variables with more than 25 occurrences). Instead it detects for each variable if this variable can be removed using Variables Elimination by Resolution (VER) without increasing the literal count. If this is the case, it eliminates the variable by VER.*

2. *So, NiVER is different from [53] which focuses on variables with two occurrences and from 2clsVER [54], which uses VER.*

3. *When VER removes a variable, a lot of resolvents must be added. The tautologies are not considered and the resolvents which are not tautologies are added to the CNF and the clauses that are containing the eliminated variable are deleted from the CNF.*

4. *NiVER does not use any other simplification such as subsumption or unit propagation (either explicitly nor implicitly) in contrast to previous preprocessors such as HyPre [50] and 2-simplify [55].*

2.9.3 SATELITE : Effective Preprocessing in SAT through Variable and Clause Elimination

The authors in [48] extend the preprocessor NiVER [49] by introducing three techniques: subsumption, self-subsuming resolution and variable elimination by substitution. These techniques could shorten the given CNF better than the previous preprocessing, as well as reducing the solving time of SAT solvers on industrial SAT instances.

Definition 72. *(Elimination by Clause Distribution)[48]*
Let Σ be a CNF, $x \in var(\Sigma)$, S_x denote the set of clauses containing x and $S_{\neg x}$ denote the set of clauses containing $\neg x$.
$S_x \otimes S_{\neg x} = \{\eta[x, C_1, C_2] : C_1 \in S_x, C_2 \in S_{\neg x}\}$ where $\eta[x, C_1, C_2]$ denotes the resolvent of the clauses C_1 and C_2 (see section 1.3.3).
If $S = S_x \cup S_{\neg x}$ then the of replacement S by $S_x \otimes S_{\neg x}$ is called elimination by clause distribution.

The authors in [48] take only the non-tautological clauses that result from the elimination process by clause distribution and discard tautologies.

Definition 73. *(Self-Subsuming Resolution)[48]*
Let Σ be a CNF and $C_1, C_2 \in \Sigma$ such that $C_1 = \{x\} \cup D_1$ and $C_2 = \{\neg x\} \cup D_2$ and $D_2 \subseteq D_1$, the process of adding the resolvent $\eta[x, C_1, C_2] = D_1 \cup D_2 = D_1$ [which subsumes C_1] and removing C_1 is called self-subsuming resolution.

See example 3 for an illustration of self-subsuming resolution operation.
To give the definition of the operation of variable elimination by substitution, we need the definition of the output of an AND gate (see section 2.9.4 for a general approach on the equations of gates).

Definition 74. *(Output of an AND Gate) [48]*
Let Σ be a CNF and $\{x, \neg a, \neg b\}, \{\neg x, a\}, \{\neg x, b\}$ be clauses in Σ. We can get from these clauses the gate (AND gate) $x = a \wedge b$ with input variables a, b and the output variable x. The output variable x is said to be functionally dependent on a, b and the equation $x = a \wedge b$ is called a definition of x.

Theorem 4. *[48]*
Let Σ be a CNF formula, $G \subseteq \Sigma$ be the subset of clauses used to recover the gate with output x. We note S_x and $S_{\neg x}$ the sets of clauses of Σ that contain x and $\neg x$ respectively. Similarly, we note G_x and $G_{\neg x}$ the set of clauses of G that contain x and $\neg x$, respectively. Let $R = S \setminus G$ the set of remaining clauses not used to recover the gate, and R_x, $R_{\neg x}$ the sets of clauses that contain x and $\neg x$, respectively. As $S = (G_x \cup R_x) \cup (G_{\neg x} \cup R_{\neg x})$, then

1. *the set of resolvents $S' = S_x \otimes S_{\neg x}$ can be partioned as follows: $S' = S'' \cup G' \cup R'$ where $S'' = (R_x \otimes G_{\neg x}) \cup (G_x \otimes R_{\neg x})$, $G' = G_x \otimes G_{\neg x}$, $R' = R_x \otimes R_{\neg x}$, and,*
2. *$S'' \models G' \cup R'$.*

Definition 75. *(Variable Elimination by Substitution)[48]*
Using the same notation as theorem 4, the process elimination of functionally dependent variables such that the clauses R' and G' are not added is called variable elimination by substitution.

In addition to these three rules, the authors in [48] used one supplementary rule which they called hyper-unary resolution. Recall that in the hyper-binary resolution (see definition 67), we take one n-ary clause and $n-1$ binary clauses and output a binary clause. The hyper-unary resolution rule is similar to hyper-binary resolution but the output is a unit clause instead of a binary clause.

Definition 76. *(Hyper-Unary Resolution)[48]*
Let C be a clause and $C_1, C_2, \ldots, C_n$ be n binary clauses, the hyper-unary resolution rule is a hyper resolution rule that takes $C, C_1, C_2, \ldots, C_n$ as input and output a unit clause.

Example 7. *Let* $\{a, b, c, d\}, \{\neg a, e\}, \{\neg b, e\}, \{\neg c, e\}, \{\neg d, e\}$ *be clauses in a CNF. The hyper-unary resolution yields the unit clause* $\{e\}$.

Using these rules, the authors in [48] provide an implementation that led to a considerable reduction of the runtime of SAT solvers in comparison with the previous preprocessors.

2.9.4 Structural Knowledge Simplification Approach

The authors in [56] extended the variable elimination by substitution rule that were presented in [48], see definition 75, by presenting a set of equations $y = f(x_1, x_2, \ldots, x_n)$ where f is one of the operators $\wedge, \vee, \Leftrightarrow$ and $y, x_1, x_2, \ldots, x_n$ are some of the variables of the given CNF.
Their preprocessor is implemented by distinguishing the input CNF into two parts

1. The first part is a set of clauses of the CNF that can be used to extract a set of equations (a set of gates) of the form $y = f(x_1, x_2, \ldots, x_n)$, $f \in \{\wedge, \vee, \Leftrightarrow\}$, where y is called the output variable of the gate.

   If the first part contains k output variables then we can reduce the set of interpretations to be checked from 2^n into 2^{n-k}, where n is the number of variables of the original formula.

2. The second part is a set of clauses of the CNF that cannot be used to extract the set of equations. A clause $C = x_1 \vee x_2 \cdots \vee x_n$ from the second part can be interpreted as a gate $TRUE = \vee(x_1, x_2, \ldots, x_n)$.
   The authors in [56] search to simplify this remaining set of clauses by reducing the number of clauses and variables involved in these clauses. Such simplifications allow the number of input variables of the extracted set of equations to be decreased.

   The simplifications of the remaining set of clauses are carried as follows.

   Remark 12. *(Simplifications of the remaining set of clauses)*

   (a) Looking for nf-blocked clauses (see definition 82) which generalize the blocked clauses notion (see definition 81) and removing these nf-blocked clauses from the given CNF.

   (b) Looking for UP-redundant clauses (see definition 83) which are special cases of redundant clauses, and removing these UP-redundant clauses from the given CNF.

   (c) Looking for subsuming resolvents (see definition 84) which subsume at least one of the clauses from the given CNF and removing these subsumed clauses from the given CNF.

   Interestingly, removing these clauses leads also to significant reductions in the number of the variables.

Definition 77. *(Set of gates [System of equations])[56]*
A set of equations is a set of gates of the form $y = f(x_1, x_2, \ldots, x_n)$ *where* $f \in \{\wedge, \vee, \Leftrightarrow\}$ *and* $y, x_1, x_2, \ldots, x_n$ *are Boolean variables. The variables* $x_1, x_2, \ldots, x_n$ *are the input variables of the equation and y is the output variable of the equation.*

Definition 78. *(The satisfiability of an equation])[56]*
An equation is satisfiable if and only if there is an assignment of a truth value to its input and output variables such that the left and right hand sides of the equation are simultaneously TRUE or simultaneously FALSE.

Definition 79. *(A model of a set of equations) [56]*
An interpretation (an assignment of a truth value to their input and output variables) of a set of equations is a model of this set if and only if it satisfies all the equations of the set.

Definition 80. *(An output variable and an Input variable of a set of gates)[56]*
A variable is an output variable of a set of gates if and only if it is an output variable of at least one gate in the set. An input variable of a set of gates is an input variable of a gate which is not an output variable of the set of gates.

An output variable of an equation is called sometimes definable variable and the equation is called a definition of this output variable. In spite of knowing k output variables, one can reduce the set of interpretations to be checked from 2^n into 2^{n-k}. However, in the general case, proving that an equation is entailed by a given CNF is coNP-complete [57]. Since the values of the output variables of the set of equations are known from the values of the input variables of this set of equations, the DPLL-like algorithms can limit the enumeration process on the input variables. The following two properties are useful for extracting a gate from a given CNF.

Property 4. *[56]*
Let Σ be a set of gates, $B \subset \Sigma$ a set of (equivalence, and,or) gates, $b \in B$ such that its output variable y occurs only in B and Σ' the set of gates obtained by the substitution of y with its definition and removing b from Σ, then Σ is satisfiable if and only if Σ' is satisfiable.

Property 5. *[56]*
Let Σ be a set of gates, any (equivalence, and, or) gate of Σ containing a literal which does not occur elsewhere in Σ, can be removed from Σ without loss of satisfiability.

We now present the notions and properties that used in [56] to simplify the remaining set of clauses.

Definition 81. *(Blocked clause [58])*
Let Σ be a CNF and $C \in \Sigma$ be a clause. C is blocked clause if and only if there is a literal $l \in C$ such that for all $C' \in \Sigma$ with $\neg l \in C'$, $C \otimes C'$ is tautological.

Property 6. *[58]*
Let C is a blocked clause in a CNF Σ, Σ is satisfiable if and only if $\Sigma \setminus \{C\}$ is satisfiable.

Definition 82. *(nf-blocked clause) [56]*
Let Σ be a CNF and $C \in \Sigma$ be a clause. C is nf-blocked clause if and only if there is a literal $l \in C$ such that there exists no resolvent in l or all resolvents are not fundamental.

Property 7. *(nf-blocked clause) [56]*
Let C is a nf-blocked clause in a CNF Σ, Σ is satisfiable if and only if $\Sigma \setminus \{C\}$ is satisfiable.

Corollary 2. *(nf-blocked clause) [56]*
Blocked clauses and clauses containing a pure literal are nf-blocked.

Definition 83. *(UP-redundant clause)[56]*
Let Σ be a CNF and $C \in \Sigma$ be a clause. C is UP-redundant clause if and only if $\Sigma \setminus \{C\} \vdash^* C$.

Definition 84. *(Subsuming resolvent) [56]*
Let Σ be a CNF, a subsuming resolvent is a resolvent from two clauses from Σ that subsumes at least one clause of Σ.

By using these removable clauses (see remark 12) one can simplify the given CNF [56]. Interestingly, by removing these clauses,the number of variables is also reduced [56].

2.9.5 ReVivAl : Reprocessing based on Vivification Algorithm

There are two kinds of preprocessors: the first category aims to eliminate variables by a limited application of resolution.The second category aims to modify the given CNF by adding and/or removing clauses. But the problem of the second approach is to measure the relation among the added and removed clauses with the resolution step. Eliminating clauses may lead to harder sub-formulas and adding clauses may

increasing the space complexity, but on the other hand the learning scheme showed that adding redundant information may be of great importance for practical instances of SAT solving. So, in [59], the authors have adopted an approach that aims to substitute the existing clauses by more constrained ones. More precisely, the vivification process consists of

1. Finding the minimal redundant sub-clauses for each redundant clause;

2. Using classical learning to get more constrained new clauses;

3. Trying to eliminate some literals from non-redundant clauses or from minimally redundant clauses.

In fact, to reduce a clause, it is first removed from the CNF formula and the opposites of its literals are assigned one by one according to their lexicographic ordering while applying unit propagation. Given a CNF formula Σ and $c = \{l_1, l_2, \ldots, l_n\}$ a clause from Σ, assuming that the order in which the literals are assigned is $(\neg l_1, \ldots, \neg l_n)$, we have two possible cases:

1. $\exists i \in \{1, \ldots, n-1\}$ s.t. $\Sigma \backslash \{c\} \cup \{\neg l_1, \ldots, \neg l_i\} \vDash^* \bot$. In this case, we have $\Sigma \backslash \{c\} \vDash^* c'$ with $c' = (l_1 \vee \cdots \vee l_i)$
   This new clause c' strictly subsumes c. Hence, the original clause can be substituted by the new one. c' is not minimally redundant modulo UP since another ordering on the literals $\{l_1, l_2, \ldots, l_i\}$ might lead to an even shorter sub-clause. But by the classical learning, the deduced sub-clause c' can be shortened again leading to an even smaller sub-clause. A new clause η can be generated by a *complete* traversal of the implication graph associated to Σ and to the assignments of the literals $\{\neg l_1, \ldots, \neg l_i\}$. The complete traversal of the implication graph ensures that the clause η contains only literals from c'. So, η is a sub-clause of $(l_1 \vee \cdots \vee l_i)$.

2. Otherwise, since unit propagation is performed after each assignment, if one of the remaining literals is assigned by this filtering operation, then a sub-clause is produced. If this is the case then the propagated literal is either assigned positively (it satisfies the removed clause of the CNF formula) or negatively (it is falsified in this clause). Considering i and j with $1 \leq i < j \leq n$, the two possible cases are:

   - $\Sigma \backslash \{c\} \cup \{\neg l_1, \ldots, \neg l_i\} \vDash^* \neg l_j$
     In this case, one can deduce: $\Sigma \backslash \{c\} \vDash^* (l_1 \vee \cdots \vee l_i \vee \neg l_j)$
     Applying resolution between this new clause and c (using the variable l_j), one obtains:
     $(l_1 \vee \cdots \vee l_j \vee \cdots \vee l_n) \otimes_R (l_1 \vee \cdots \vee l_i \vee \neg l_j) = (l_1 \vee \cdots \vee l_{j-1} \vee l_{j+1} \vee \cdots \vee l_n)$. This new clause clearly subsumes c. Hence, the original clause can be substituted by the new deduced one.
   - $\Sigma \backslash \{c\} \cup \{\neg l_1, \ldots, \neg l_i\} \vDash^* l_j$
     In this case, one can deduce: $\Sigma \backslash \{c\} \vDash^* (l_1 \vee \cdots \vee l_i \vee l_j)$
     In this case too, the produced clause subsumes c and enables to "remove" literals from it.

Hence, briefly, we get the following:

1. The iterative assignments of the opposite literals of a clause can produce a reduced clause.

2. Some assignments might lead to a conflict. So, the conflict analysis can be used to produce smaller sub-clauses in a polynomial time.

3. Hence by above rules and learning scheme, the given CNF can be *vivified*, that is, it will be easier to solve.

The Vivification preprocessing is described in Algorithm 4.

Algorithm 4: Vivification of a CNF formula [59]

Input: Σ : a **CNF formula**
Output: a vivified CNF formula

```
1  change ⟵ TRUE; ;
2  while change do
3      change ⟵ FALSE ;
4      foreach c ∈ Σ do
5          Σ ⟵ Σ\{c} ;    Σ_b ⟵ Σ ;
6          c_b ⟵ ∅ ;    shortened ⟵ FALSE ;
7          while (Not(shortened) And (c ≠ c_b)) do
8              l ⟵ select_a_literal(c\c_b) ;
9              c_b ⟵ c_b ∪ {l} ; Σ_b ⟵ (Σ_b ∪ {¬l}) ;
10             if ⊥ ∈ UP(Σ_b) then
11                 c_l ⟵ conflict_analyse_and_learn() ;
12                 if c_l ⊂ c then
13                     Σ ⟵ Σ ∪ {c_l} ;
14                     shortened ⟵ TRUE ;
15                 else
16                     if |c_l| < |c| then
17                         Σ ⟵ Σ ∪ c_l ; c_b ⟵ c ;
18                     if c ≠ c_b then
19                         Σ ⟵ Σ ∪ {c_b} ;
20                         shortened ⟵ TRUE ;
21             else
22                 if ∃(l_s ∈ (c\c_b)) s.t. l_s ∈ UP(Σ_b) then
23                     if (c\c_b) ≠ {l_s} then
24                         Σ ⟵ Σ ∪ {c_b ∪ {l_s}} ;
25                         shortened ⟵ TRUE ;
26                 if ∃(l_s ∈ (c\c_b)) s.t. ¬l_s ∈ UP(Σ_b) then
27                     Σ ⟵ Σ ∪ {c\{l_s}} ;
28                     shortened ⟵ TRUE ;
29             if Not(shortened) then Σ ⟵ Σ ∪ {c} ;
30             else change ⟵ TRUE ;
31     return Σ ;
```

2.10 CDCL-based SAT Algorithm

In this section we introduce a generic form of CDCL-based SAT algorithm 5 (see [34]).
This algorithm works as follows:

1. The decision literals set and learnt clauses database are first initialized to an empty set (lines 1 and 2).

2. The closure of the current CNF Σ using unit propagation (denoted by Σ^*) is computed at each iteration (line 5).

3. If the algorithm detects a conflict (lines 6-10) then we have two cases:

   - The set of decisions literals is empty (line 6), in this case the algorithm returns unsatisfiable

Algorithm 5: CDCL-based SAT solver [34]

Input: CNF formula Σ

Output: A model of Σ or unsat if Σ is unsatisfiable

```
1  D ← ∅;                                              /* decision literals */
2  Δ ← ∅;                                          /* learnt clauses database */
3  while (TRUE) do
4      S ← Σ ∧ Δ ∧ D;
5      if (S* = ⊥) then
6          if ((D = ⟨⟩) then return unsat;
7          α ← learningFromConflict(S) ;
8          m ← assertion level of α;
9          D ← D^m;
10         Δ ← Δ ∧ α ;
11     else
12         if (timeToReduce()) then
13             Δ ← reduceLearntDB();
14         if (timeToRestart()) then
15             D ← ∅;
16             S ← Σ ∧ Δ ∧ D;
17         ℓ ←decide();
18         if (ℓ = null) then
19             return D;
20         D ← D ∧ ℓ;
```

- The set of decisions literals is not empty, in this case a new asserting clause is derived by learning from conflict (line 7), then the algorithm :
  - Backtracks to the assertion level m (line 9);
  - Adds the asserting clause to the learnt clauses database (line 10).

4. If there is no conflict (lines 12-20) then the algorithm performs one or both of the following two steps .
   - Reduces the learnt database (lines 12-13) using reduction.
   - Restarts the search process (lines 14-16) using restart policies.
5. Choose a new decision literal using VSIDS branching rule and polarity functions [60] then
   - Add this literal to the set of decisions (line 20).
   - Give it its associated level (line 20).
6. A model is found and the algorithm returns satisfiable if all the variables are assigned (lines 18-19).

Part III

Polynomial Fragments and UP-based Polynomial Fragments of SAT

Chapter 3

Tractable Classes and Hierarchies of Tractable Classes

Tractable classes are specific fragments of SAT whose instances can be solved in polynomial time. Most (but not all) of these tractable classes are provided with two polynomial time algorithms. One algorithm characterizes the class (it checks whether or not the given instance of SAT belongs to this class) and, when the instance belongs to the class, the second algorithm solves it by deciding whether this given instance of SAT is satisfiable or not. There are many tractable subclasses of SAT; we present in this chapter the well-known classes and focus on tractable classes that make use of the UP algorithm in some direct or more elaborate ways; we will extend some of them in the forthcoming chapters to yield new tractable classes. Notice that it might be believed that the number of tractable classes is small following a wrong interpretation of Schaefer's dichotomy theorem [61]: this is not the case and there are many tractable classes. Actually, Schaefer's theorem is for classes that can be recognized in log space.

3.1 Preliminaries and Notations

We need some notions and notations in order to introduce the tractable classes smoothly.

Definition 85. *(Graph,Walk,Trail and Path)*

1. *(Graph)*

   - *A (finite) undirected graph is an ordered pair $G = (V, E)$ where V is a (finite) set of vertices (nodes or points) and E is a (finite) set of edges (arcs or lines), where an unordered pair $\{u, v\}$ is an edge in G where $u, v \in V$ if $\{u, v\} \in E$. Abstractly (from a set-theoretic perspective) a finite undirected graph is a subset of the set $\{\{u, v\} : u, v \in V, V$ is a finite set$\}$.*

   - *A (finite) directed graph is an ordered pair $G = (V, E)$ where V is a (finite) set of vertices (nodes or points) and E is a (finite)e set of arcs (directed edges), where an ordered pair (u, v) is an edge in G where $u, v \in V$ if $(u, v) \in E$. Abstractly (from a set-theoretic perspective) a finite directed graph is a binary relation that is a subset of the set $\{(u, v) : u, v \in V, V$ is a finite set$\} = V \times V$ where $V \times U$ denotes the Cartesian product of the two sets V and U.*

2. *(Walk).*
   A walk in a directed graph is a sequence $v_0, e_1, v_1, \ldots, e_n, v_n$ where v_i, $i = 1, \ldots, n$ are vertices, and e_i is an edge from v_{i-1} to v_i, $i = 1, \ldots, n$.

3. *(Trail).*
   A trail is a walk in which all edges are distinct.

4. *(Path).*
   A path is a trail in which all vertices (except possibly the first and last ones) are distinct.

Definition 86. *(Tree, Rooted tree, Labeled tree, Labeled rooted tree and Depth of a node)*

1. *(Tree).*
   A tree is an undirected graph where any two vertices in it can be connected by a unique simple path. G is a tree if and only if G is connected and has n-1 edges.

2. *(Rooted tree).*
   A rooted tree is a tree with one root vertex.

3. *(Labeled tree).*
   A labeled tree is a tree in which each vertex is given a unique label.

4. *(Labeled rooted tree).*
   A labeled rooted tree is a rooted and labeled tree.

5. *(Depth of a node).*
   The depth of a node is the number of arcs in the path that connect the node to the root.

Definition 87. *(Directed graph and directed walk)*

1. *(Directed graph).*
   A directed graph (or digraph) is a graph where the edges have a direction associated with them.

2. *(Directed walk).*
   A directed walk $v_0 \to \cdots \to v_n$ in a directed graph is a sequence $v_0, e_1, v_1, \ldots, e_n, v_n$ where v_i, $i = 1, \ldots, n$ are vertices, and e_i is a directed edge from v_{i-1} to $v_i, i = 1, \ldots, n$.

Definition 88. *(Directed acyclic graph) A directed acyclic graph is a directed graph without directed cycles (i.e., there is no directed walk from a vertex that loops back to the same vertex).*

Definition 89. *(Strongly connected graph)*

1. *A graph is strongly connected if every two vertices are reachable (i.e., there is a walk from one of them to the other). The maximal strongly connected subgraphs of a graph are vertex-disjoint and are called strongly connected components. The strongly connected components of graph can be computed in $O(m + n)$ time [62] using depth-first-search.*

2. *If S_1 and S_2 are strongly connected components such that an edge leads from a vertex in S_1 to a vertex in S_2, then S_1 is a predecessor of S_2 and S_2 is a successor of S_1.*

Definition 90. *(Undirected bipartite graph)*
An undirected bipartite graph is an undirected graph whose set of vertices is divided into two sets U and V where $U \cap V = \emptyset$ and each edge in the graph has one vertex in U and the other vertex in V.

Definition 91. *(Undirected bipartite graph of a CNF.)*
Let Σ be a CNF. The undirected bipartite graph of Σ (which will be denoted by $UBG(\Sigma)$) is the undirected bipartite graph where $U = C(\Sigma)$ and $V = V(\Sigma)$. Each clause $C \in U$ is connected with edges to the variables involved in it. Also we will denote the set of edges of $UBG(\Sigma)$ by $EDG(UBG(\Sigma))$.

Definition 92. *(Hypergraph)*
A hypergraph H is a pair $H = (X, E)$ where X is a set (called nodes or vertices), and $E \subseteq X$ is a non-empty set (called hyperedges or edges).

Definition 93. *(Ordered relation)*
An ordered relation is a binary relation $\leq$ over a given set S which is reflexive, antisymmetric, and transitive, that is, it satisfies :
$a \leq a \; \forall a \in S$; (reflexivity)
If $a \leq b$ and $b \leq a$, then $a = b \; \forall a, b \in S$ (antisymmetry);
If $a \leq b$ and $b \leq c$, then $a \leq c \; \forall a, b, c \in S$ (transitivity).

3.2 Trivial tractable classes

We present in this section some trivial tractable classes.

3.2.1 Monotone, positive and negative formulas

Definition 94. *A CNF is monotone if and only if all its variables are pure (appears positively or negatively) in the CNF.*

Finding a satisfiable truth assignment for monotone CNFs is straightforward (just let the positive pure variables take the value TRUE and the negative pure variables take the value FALSE).

Definition 95. *A CNF is positive (negative) if and only if all its variables is appears positively (negatively) in the CNF. A positive (negative) CNFs is a special case of a monotone CNFs.*

3.2.2 No-positive clause and No-negative clause formulas

Definition 96. *A CNF is No-positive clause formula if and only if it has no positive clause and it is No-negative clause formula if and only if it has no negative clause.*

Finding a satisfiable truth assignment for No-positive clause CNFs is trivial, just select a negative literal from every clause and assign the value TRUE to it (FALSE to its corresponding variable) and similarly for No-negative clause CNFs, just select a positive literal from every clause and assign the value TRUE to it.

3.2.3 1-valid and 0-valid formulas

Definition 97. *(clause-variable matrix).*
Let Σ be a CNF. Its clause-variable matrix, denoted by M_Σ is the matrix such that its elements (i, j) have the value $+1$ if the clause C_i has the literal a_j, -1 if it has the literal $\neg a_j$ and 0 otherwise, where $a_j \in V(\Sigma)$.

Definition 98. *(1-valid and 0-valid formulas).*
A formula is 1-valid (resp. 0-valid) if and only if each column of its clause-variable matrix contains at least $+1$ (resp. -1).

A 1-valid (respectively 0-valid) has a trivial solution that can be obtained by just assigning all the variables to TRUE (respectively FALSE).

3.3 Well-known Tractable Classes

We present in this section the well-known tractable classes where the UP-technique plays a central role.

3.3.1 2SAT formulas

A clause is called a Krom clause or 2-clause if and only if it has at most two literals (i.e., it is either the empty clause or a unit clause or a binary clause). A Krom CNF (often called 2CNF or 2SAT) is a set of Krom clauses.

The recognition of a 2CNF formula is trivial and can be done in linear time by checking the length of the clauses in the given CNF.

For solving 2CNF there are many algorithms.

First, the number of Krom clauses depends on the number of variables (the number of Krom clauses is $4n^2 + n + 1$ if the number of variables is n). Hence the DP algorithm takes at most $O(n^2)$ space and solves the satisfiability problem for 2CNF in polynomial time [63].
There are many algorithms that solve 2CNF in linear Time [64, 65, 66].
We overview two well-known algorithms: one has polynomial time while the other has linear time complexity.

Polynomial-Time Algorithm Based on Davis-Putnam [24]

When two resolvable clauses belong to 2SAT then their resolvent also belongs to 2SAT. Consequently, if we add resolvents to 2-SAT instance Σ with n variables, then we get at most $1 + 2n + 4\binom{n}{2} = 2n^2 + 1$ clauses. Hence the algorithm terminates by adding at most $O(n^2)$ resolvents. If we do not get the empty clause then Σ is satisfiable; else (i.e., if we get the empty clause), Σ is unsatisfiable.

A Linear-Time Algorithm [65, 38]

We describe the algorithm presented in [65], which runs in linear-time.
Let Σ be a 2-SAT CNF with m clauses and n variables $\{x_1, \ldots, x_n\}$. Σ is associated to a directed graph $G(\Sigma)$ defined as follows.

1. For each variable x_i in Σ, $x_i$and $\neg x_i$ are vertices of the graph $G(\Sigma)$.
2. For each clause $x_i \vee x_j$ in Σ, the edges $\neg x_i \to x_j$ and $\neg x_j \to x_i$ are added to the graph $G(\Sigma)$.

Hence, $G(\Sigma)$ has vertices $\{x_1, \ldots, x_n, \neg x_1, \ldots, \neg x_n\}$ and directed edges $\{(\neg x_i, x_j), (\neg x_j, x_i) | \{x_i, x_j\} \in \Sigma\}$. So, $G(\Sigma)$ has $2n$ vertices and $2m$ edges.

Remark 13. *Let Σ be a CNF and let $u \rightsquigarrow v$ denote a directed walk $u \to \cdots \to v$ in the graph $G(\Sigma)$.*

1. *If $u \rightsquigarrow v$ then if u has the value TRUE in a satisfying truth assignment, then v also has the value TRUE in this satisfying truth assignment.*
2. *If $u \rightsquigarrow \neg u$ then u has the value FALSE in every satisfying truth assignments.*
3. *If $u \rightsquigarrow v \rightsquigarrow u$ then u and v has the same truth value in every satisfying truth assignment.*
4. *$u \rightsquigarrow v$ if and only if $\neg v \rightsquigarrow \neg u$.*

Property 8. *[65].*
A 2-SAT formula Σ is unsatisfiable if and only if $G(\Sigma)$ contains a directed walk $x \rightsquigarrow \neg x \rightsquigarrow x$.

Algorithm 6: Linear time 2-SAT algorithm

```
Input 2-SAT CNF Σ;
Output Σ is SAT or UNSAT;
S ← strongly connected components of G(Σ);
forall the unassigned component S in S do
    if S contains literals u and ¬u as vertices then
        return UNSAT;
    set each literal labelling vertices of S to TRUE;
    set each literal labelling vertices of S̄ to FALSE;
return SAT ;
```

Algorithm 6 finds a satisfying assignment in $O(m + n)$ time when $G(\Sigma)$ contains no directed walk of the form $x \rightsquigarrow \neg x \rightsquigarrow x$.

3.3.2 Horn, Reverse-Horn and Renamable-Horn Formulas

A Horn (respectively reverse-Horn) clause is a clause $a_1 \vee a_2 \vee \cdots \vee a_k$ such that at most one of its literals is positive (respectively negative). A HornCNF (or HornSAT) is a collection of Horn clauses.

The recognition of Horn formulas is trivial and can be done in linear time by simply checking the number of positive literals in the clauses of the given CNF.

There are several algorithms that can solve HornCNF in linear time [67, 68, 69, 2, 70, 71], some of them use the unit propagation approach.

In fact, unit propagation is complete for Horn class, since a Horn CNF without unit clauses is satisfiable (it is No-positive clause formula, see section 3.2.2). Hence if a Horn CNF has unit clauses then we can apply UP on it and we have two cases

1. If we encounter the empty clause then the given CNF is unsatisfiable.
2. If the resulting CNF (it is also a Horn CNF) has no unit clause then it (and hence the given CNF) is satisfiable.

Renamable-Horn Formulas

A renamable-Horn (or Hidden Horn) formula is a formula that can transformed to Horn formulas by replacing some of its literals by their negations and vice versa.

Definition 99. ***(Variable Complementation or Variable Renaming).***
Let Σ be a CNF with n variables $X = \{x_1, x_2, \ldots, x_n\}$ and let $S \subseteq X$, The CNF Σ^S results from Σ by renaming the variables in S, i.e., by replacing all occurrences of x_i by $\neg x_i$ and all occurrences of $\neg x_i$ by x_i.

Note that Σ^S and Σ are semantically equivalent.

Definition 100. *(Renamable-Horn)*
Σ is renamable-Horn if and only if some renaming of Σ is Horn.

We can see the renaming as a mapping $r : L(\Sigma) \rightarrow L(\Sigma)$ such that

1. $r(x) = x$ or $r(x) = \neg x, \forall x \in L(\Sigma)$.
2. $r(x) = t$ if and only if $r(\neg x) = \neg t$, $\forall x, t \in L(\Sigma)$.

There exist 2^n such mappings were n is the number of variables of the CNF Σ.

Example 8.
1. *$\Sigma = \{\{\neg x, \neg y, z\}, \{x, y\}, \{\neg z\}\}$ is renamable-Horn (replace x by $\neg x$ and vice-versa and replace z by $\neg z$ and vice-versa).*
2. *$\Sigma = \{\{\neg x, \neg y, z\}, \{x, y, \neg z\}\}$ is not renamable-Horn. Since for all the 8 renaming mappings, we cannot obtain a Horn CNF.*

The existence of this Horn renaming (the recognition algorithm) can be checked in linear time, and these algorithms provide the renaming set also. So we can solve renamable-Horn formulas in linear time: first, by renaming the formula as a Horn formulas and then by solving these Horn formulas [72, 73, 74, 75].

As an illustration, we describe two recognition algorithms: one runs in quadratic time and is due to Lewis [72] and the other exhibits a linear time and is due to Aspvall [73].

A Quadratic Time recognition Algorithm for Renamable-Horn [Lewis [72]]

Lewis [72] presented a quadratic time recognition algorithm for renamable-Horn. The idea of the algorithm is to reduce the renamable-Horn problem to the satisfiability problem of 2SAT instance.

Notation 5. *[72]*
Let $\Sigma = \{C_1, \ldots, C_m\}$ be a CNF with m clauses and n variables where $|C_i| = l_i$, $1 \leq i \leq m$ and let $C_i = \{u_{i_1}, \ldots, u_{i_{l_i}}\}$ then $\Sigma^- = \bigcup_{i=1}^{m} \bigcup_{1 \leq j < k \leq l_i} \{\{u_{i_j}, u_{i_k}\}\}$.

Property 9. *[72]*
Σ is renamable-Horn if and only if Σ^- is satisfiable.

The CNF Σ^- can be constructed in $O(mn^2)$ time [72] and tesing its satisfiability and finding a model of it can be done in linear time, see section 3.3.1.
This model can be used to rename Σ [72]. So the total time for renaming Σ is $O(mn^2)$ [72].

A linear time recognition algorithm for renamable-Horn (Aspvall [73])

The algorithm of Aspvall [73] is designed to recognize renamable-reverse-Horn formulas. This is equivalent to recognize renamable Horn formulas. Indeed, one just needs to replace positive literals by negative ones and vice versa. Such operation can be done in linear time.
The linear time is achieved by reducing the number of clauses in the associated 2SAT, see section 3.3.2. This reduction extends the set of variables [75] .

Notation 6. *[73]*

1. *Let Σ be a CNF and $x \in V(\Sigma)$, $truth_x(\Sigma)$ is the truth value of x in a satisfying assignment of Σ. If Σ is unsatisfiable, $truth_x(\Sigma)$ is undefined.*
2. *Let $C = u_1 \vee u_2 \vee \cdots \vee u_k$ be a clause in Σ and let $y_1^C, y_2^C \ldots, y_{k-1}^C$ be auxiliary variables associated with C.*
   $\Sigma^= = \bigwedge_{C \in \Sigma, |C| \geq 2} [(u_1 \vee y_1^C) \wedge [\bigwedge_{1<i<k} ((\neg y_{i-1}^C \vee u_i) \wedge (\neg y_{i-1}^C \vee y_i^C) \wedge (u_i \vee y_i^C))] \wedge (\neg y_{k-1}^C \vee u_k)]$

Property 10. *[73]*
Let Σ be a CNF and Σ^- be as defined in section 3.3.2.

1. *Σ is reverse-Horn if and only if the truth assignment that assigns the value TRUE to all variables of Σ satisfies Σ^- .*
2. *Σ is renamable-reverse-Horn if and only if Σ^- is satisfiable. moreover if Σ^- is satisfiable then the renaming mapping $r : L(\Sigma) \rightarrow L(\Sigma)$ such that*
   - *$r(x) = x$ if $truth_x(\Sigma^-) = TRUE$*
   - *$r(x) = \neg x$ if $truth_x(\Sigma^-) = FALSE$*

   Transfoms Σ to a reverse-Horn CNF.
3. *$\Sigma^=$ is satisfiable if and only if Σ^- is satisfiable*

Property 11. *[73]*
Σ is renamable-reverse-Horn if and only if $\Sigma^=$ is satisfiable. Moreover if $\Sigma^=$ is satisfiable then the renaming mapping $r : L(\Sigma) \rightarrow L(\Sigma)$ such that

- *$r(x) = x$ if $truth_x(\Sigma^=) = TRUE$*
- *$r(x) = \neg x$ if $truth_x(\Sigma^=) = FALSE$*

transforms Σ to a reverse-Horn CNF.

The CNF $\Sigma^=$ is a 2SAT formula such that $|\Sigma^=| = O(|\Sigma|)$ and the number of its variables is $O(n + |\Sigma|)$ where n is the number of variables of Σ. So using any linear time algorithm for 2SAT (see section 3.3.1), we can decide if a reverse-Horn-renaming exists and construct one in $O(n + |\Sigma|) = O(|\Sigma|)$ time. This construction is performed as follows [73]:

- Constuct $\Sigma^=$.
- Check if $\Sigma^=$ is satisfiable.
  1. If no, there is no reverse-Horn-renaming .
  2. If yes, then the reverse-Horn-renaming mapping is $r : LIT(\Sigma) \rightarrow LIT(\Sigma)$ such that
     - $r(x) = x$ if $truth_x(\Sigma^=) = TRUE$.
     - $r(x) = \neg x$ if $truth_x(\Sigma^=) = FALSE$.

3.3.3 Extended Horn, Hidden Extended Horn, Simple Extended Horn, CC-Balanced Formulas and SLUR Algorithm

Extended Horn, Hidden Extended Horn Formulas and Simple Extended Horn

The "hidden" extended Horn is an extended class of Horn that can be solved in linear time by the same technique that solves Horn class (i.e., the UP technique).
More precisely the UP-based algorithm (see algorithm 7) can be used to find a solution to the linear system $Ax \geq b$
$x \leq e$ (II)
$x \geq 0$.
which is the linear relaxation (see definition 101) of the linear system
$Ax \geq b$
$x \leq e$ (I)
$x \geq 0$, x integral.
The "hidden" extended Horn is defined in such a way that an instance in it is satisfiable if and only if algorithm 7 finds no contradiction.
So to introduce the "hidden" extended Horn class, we need the following representation of a CNF as a linear system.

Notation 7. *[76]*
Let A be an $m \times n$ matrix and x and b are vectors of lengths n and m respectively and e and 0 are the vectors of ones and zeros of length n respectively. Consider the linear system
$Ax \geq b$
$x \leq e$ *(I)*
$x \geq 0$*, x integral.*
If $x = (x_1, \ldots, x_n)$ is a solution to this system then $x_i \in \{0, 1\}, i = 1, \ldots, n$. One [76] can represent a clause as a linear inequality such that the positive literal x_1 in the clause corresponds to the binary variable x_1 in the inequality and the negative literal $\neg x_1$ in the clause corresponds the binary variable $1 - x_1$ in the inequality and such that the summation of the variables in the inequality is greater than or equal 1 where the value $x_j = 1$ in the inequality corresponds to the $x_j = TRUE$ in the clause and the value $x_j = 0$ in the inequality corresponds to the $x_j = FALSE$ in the clause.
For example the clause $x_1 \vee \neg x_2$ corresponds to the inequality $x_1 + 1 - x_2 \geq 1$, $x_1, x_2 \in \{0, 1\}$. We can transform the one in the left side of the inequality to the right side and rewrite the inequality as $x_1 - x_2 \geq 0$, $x_1, x_2 \in \{0, 1\}$. In general, one can transform the 1s numbers in the left side of the inequality to the right side and rewrite the inequality as $ax \geq a_0$ such that

1. *a is a vector where its components are in $\{0, 1, -1\}$;*
2. *x is a vector $(x_1, \ldots, x_n)$, $x_i \in \{0, 1\}, i = 1, \ldots, n$;*
3. *a_0 is 1 minus the number of -1's in a.*

So the CNF with m clauses can be represented as a linear system (I)

Definition 101. *(The linear relaxation of a linear system) [76]*
The linear relaxation of the linear system (I) is obtained by removing the integral condition on the vector x.

If the linear relaxation of a linear system has a solution then one can use an algorithm in linear programming to find this solution.
To define extended Horn class and simple extended Horn, we use elements of graph theory [77] (this is not the definition used in [76] but they are equivalent).

Definition 102. *(Extended Horn and Simple Extended Horn)[77]*
Let Σ be a CNF.

- Σ *is extended Horn if there is a rooted directed tree* T*, indexed on the variables of* Σ *and* $\forall C \in \Sigma$*,*
  1. *All the positive literals of* C *are consecutive on a single path of* T*;*
  2. *There is a partition of the negative literals of* C *into sets* $N_1, N_2, \ldots, N_{n_c}$*, where* n_c *is at least 1, but no greater than the number of negative literals of* C*, such that for all* $1 \leq i \leq n_c$*, all the variables of* N_i *are consecutive on a single path of* T*;*
  3. *For at most one* i*, the path in* T *associated with* N_i *begins at the vertex in* T *from which the path associated with positive literals begins;*
  4. *For all remaining* i*, the path in* T *associated with* N_i *begins at the root of* T*.*
- Σ *is simple extended Horn if condition 3 above is not allowed, i.e., if for all* i*, the path in* T *associated with* N_i *begins at the root of* T*.*

Property 12. *[78]*
HORN $\subset$ *Simple Extended Horn* $\subset$ *Extended Horn.*

Property 13. *[76]*
An extended Horn set is satisfiable if and only if the linear relaxation of the linear system (I) has a solution.

Algorithm 7: UP-ExtendedHorn [76]

1 If there are no unit clauses, assign every variable the value $\frac{1}{2}$, and stop. Otherwise, go to Step 2.
2 Fix the values of the variables in unit clauses so as to satisfy the unit clauses. If this requires setting a variable to both 0 and 1, stop: the linear relaxation is insolvable. Otherwise, go to Step 3.
3 Perform all possible unit resolutions on the current set of clauses, and eliminate all clauses subsumed by the unit clauses in the current set (including the unit clauses themselves). Add the resolvents to the current set, and return to Step 1.

The linear relaxation of (I) does not have a solution if in the algorithm 7 a contradiction is produced in step 2, but if the algorithm terminates in step 1, then since all remaining clauses have at least two literals, they are satisfied by giving every variable in them the value $\frac{1}{2}$. So, we have

Property 14. *[76]*
Linear relaxation of (I) has a solution if and only if algorithm 7 finds a solution.

Property 15. *[76]*
An explicit or hidden extended Horn set is satisfiable if and only if algorithm 7 finds no contradiction.

Unfortunately, there is no known algorithm to recognize extended Horn and hidden extended Horn class, but fortunately there is a recognition algorithm for simple extended Horn.

CC-Balanced (or Balanced) Formulas

Definition 103. *(CC-Balanced (or Balanced) Formulas)[79]*
A CNF Σ *is CC-balanced if and only if for every submatrix of* M_Σ *that has exactly two nonzero entries for each row and column, the sum of the entries is a multiple of 4.*

Example 9. $\Sigma = \{\{x_1, \neg x_2\}, \{\neg x_1, x_2\}\}$ *is a balanced formula where* $M_\Sigma = \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$*, note that the sum of the entries is 0 and hence a multiple of 4.*

CC-balanced CNF can be recognized and solved in polynomial time [80].

A decision algorithm for CC-balanced formulas

We need the following definitions for the solving algorithm.

Definition 104. *(Generalized Set Covering Polytope) [80]*
Let M be $0, \pm 1$ matrix and let $n(M)$ be the vector whose ith components $n_i(M)$ are the number of -1's in the ith row of matrix M.
The generalized set covering polytope of M is $Q(M) = \{x : Mx \geq \mathbf{1} - n(M), \mathbf{0} \leq x \leq \mathbf{1}\}$.

Definition 105. *(ideal $0, \pm 1$ matrix)[80]*
A $0, \pm 1$ matrix is ideal if and only if the generalized set covering polytope is integral.

Theorem 5. *[80]*
Let M be $0, \pm 1$ matrix, the following statements are equivalent:

1. *M is balanced.*
2. *For each submatrix A of M, the generalized set covering polytope $Q(A)$ is integral.*

Property 16. *[80]*
Let Σ be a CNF with m clauses $C_i = (\bigvee_{j \in P_i} x_j) \vee (\bigvee_{j \in N_i} \neg x_j)$, $i = 1, \ldots, m$. Σ is satisfiable by a truth assignment if and only if the corresponding 0,1 vector satisfies the system of inequalities
$\sum_{j \in P_i} x_j - \sum_{j \in N_i} x_j \geq 1 - |N_i|$, $i = 1, \ldots, m$.

Notes 12. *[80]*
We can rewrite the system of inequalities in Property 16 as
$Mx \geq \mathbf{1} - n(M)$.....(III)
where M is $0, \pm 1$ matrix.

Definition 106. *(Ideal formulas) [80]*
A CNF is ideal if and only if the corresponding $0, \pm 1$ matrix M in (III) is ideal.

Property 17. *[80]*
Every balanced CNF is an ideal CNF.

We can solve the satisfiability problem of the ideal class (and hence of the balanced class) in polynomial time by linear programming [80]. But using the following property, we can solve the satisfiability problem of the ideal class (and hence of the balanced class) by a more efficient algorithm than linear programming [80] (see Algorithm 8).

Property 18. *[80]*
Let Σ be an ideal CNF without unit clauses. For every variable x in Σ there exist at least two truth assignments satisfying Σ: one in which x is TRUE and one in which x is FALSE.

A recognition algorithm for CC-balanced formulas
The recognition algorithm for CC-balanced formulas is somehow complicated. We introduce it briefly, to do that we need some definitions from graph theory (for details see [80]).

Definition 107. *[80]*

1. *(chord in a cycle)*
   A chord is an edge that is not part of the cycle but connects two vertices (that are necessarily not adjacent) of the cycle.
2. *(A hole in an undirected graph)*
   In an undirected graph, a hole is a chordless cycle of length greater than 3.
3. *(A balanced cycle)*
   A cycle is balanced if and only if its length is a multiple of 4.

Algorithm 8: ideal-balanced decision algorithm[80]

```
Input: An ideal CNF formula Σ;
Ouput: Either the CNF formula Σ is SAT or it is UNSAT;
if Σ = {} then
    return SAT;
while Σ ≠ {} do
    Σ ← UP(Σ);
    if ⊥ ∈ Σ then
        return UNSAT;
    Choose arbitrary clause C in Σ;
    Choose arbitrary variable x in C;
    Choose arbitrary one of the following;
    Σ ← Σ ∧ x;
    Σ ← Σ ∧ ¬x;
return SAT;
```

4. *(A balanced graph)*
   A graph is balanced if and only if all its chordless cycles are balanced.

5. *(A major node for a hole)*
   A node v is major for a hole H if and only if v has at least three neighbors in H.

Notes 13. *[80])*
A balanced graph is simple and bipartite.

Definition 108. *(The bipartite representation of a matrix)[80]*

1. *(The bipartite representation of a 0,1 matrix)*
   The bipartite representation of a 0, 1 matrix A is the bipartite graph $G(A) = (V^r \cup V^c, E)$ *having a node in* V^r *for every row of A, a node in* V^c *for every column of A and an edge* ij *joining nodes* $i \in V^r$ *and* $j \in V^c$ *if and only if the entry* a_{ij} *of A equals 1.*

2. *(bipartite representation of a 0,*± 1 *matrix)*
   The bipartite representation of a 0, ± 1 *matrix A is the signed bipartite graph* $G(A) = (V^r \cup V^c, E)$ *having a node in* V^r *for every row of A, a node in* V^c *for every column of A and an edge* ij *joining nodes* $i \in V^r$ *and* $j \in V^c$ *if and only if the entry* a_{ij} *of A is nonzero. Moreover,* a_{ij} *is the sign of the edge* ij.

Notes 14. *[80]*

1. *The bipartite representation of a 0,*± 1 *matrix extends the bipartite representation of a 0,1 matrix.*

2. *For any bipartite graph* $G(A) = (V^r \cup V^c, E)$*, with signs* ± 1 *on its edges, there is a unique matrix A, for which* $G = G(A)$ *(up to transposition of the matrix, permutation of rows and permutation of columns).*

Definition 109. *(A clean signed bipartite graph) [80]*
A signed bipartite graph is clean if and only if it is either balanced or contains a smallest unbalanced hole H with no major vertices for H.

Theorem 6. *[80]*
Let H be the smallest unbalanced hole in a signed bipartite graph. H contains two edges such that every major node for H is adjacent to at least one of the endnodes of these two edges.

Property 19. *[80]*
There exists a polynomial time algorithm which takes as input a signed bipartite graph G and outputs a clean graph G′, such that G is balanced if and only if G′ is balanced.

Definition 110. *(A wheel and an odd wheel in bipartite graph) [80]*
In a bipartite graph, a wheel is (H, v) where H is a hole and v is a node having at least three neighbors in H. The wheel (H, v) is odd if and only if v has an odd number of neighbors in H .

Definition 111. *(A detectable 3-wheel) [80]*
A detectable 3-wheel is a wheel (H, v) where v has three neighbors in H and two of the neighbors of v in H have distance two in H .

Definition 112. *[80]*

1. *(3-path configuration)*
   In a bipartite graph, a 3-path configuration is an induced subgraph consisting of three internally node-disjoint paths connecting two nonadjacent nodes u and v and containing no edge other than those of the paths.

2. *(3-odd-path configuration)*
   In a bipartite graph, if the three paths in the 3-path configuration have an odd number of edges, the 3-path configuration is called a 3-odd-path configuration.

Theorem 7. *[80]*
A bipartite graph is balanceable if and only if it does not contain an odd wheel or a 3-odd-path configuration as an induced subgraph.

Property 20. *(Smallest 3-odd-path configuration in subgraph from smallest 3-odd-path configuration in graph)[80, 81]*
In a bipartite graph G, consider a 3-odd-path configuration with the smallest number of nodes, induced by paths P_1, P_2, P_3 connecting nodes u and v and let m_i be a middle node of path P_i, $i = 1, 2, 3$. In a subgraph obtained from G by removing some neighbors of u and v, any shortest path from m_i to u and v can be substituted for P_i yielding another smallest 3-odd-path configuration.

Property 21. *[81]*
Using property 20 Zambelli [81] show that

1. *There exists a polynomial time algorithm to detect whether a bipartite graph contains a 3-odd-path configuration.*

2. *There exists a polynomial time algorithm that checks whether a bipartite graph that does not contain a 3-odd-path configuration contains a detectable 3-wheel.*

But by Theorem 7, if a bipartite graph contains a 3-odd-path configuration or a detectable 3-wheel then it is not balanceable.

Property 22. *[81]*
Let G be a clean signed bipartite graph that does not contain a 3-odd-path configuration or a detectable 3-wheel. There exists a polynomial time algorithm that checks whether G is balanced .

SLUR algorithm

The structural properties of CNFs were not use to define the class *SLUR* (Single Look-ahead Unit Resolution solvable) but a nondeterministic algorithm was used instead.

Definition 113. *(SLUR formulas)[82, 83]*
A function $F \in SLUR$ if and only if the algorithm 9 does not return "give up" for any of the nondeterministic choices in steps 7 and 15

The SLUR class contains both hidden extended Horn and balanced formulas. In fact, the SLUR algorithm solves a larger class than both hidden extended Horn and balanced classes [83].

Property 23. *[83]*
Let Σ and Σ' be CNFs such that Σ is solved by the SLUR algorithm and the clauses of Σ' are logical consequences of Σ then the SLUR algorithm solves $\Sigma \cup \Sigma'$.

Example 10. *[83]*
Let Σ be a CNF and $C_1, C_2, C_3, C_4 \in \Sigma$ and let a, b be variables such that $\{a, b\} \subseteq C_1$, $\{a, \neg b\} \subseteq C_2$, $\{\neg a, b\} \subseteq C_3$, $\{\neg a, \neg b\} \subseteq C_4$: we have that Σ does not belong to the hidden extended Horn class.

Example 11. *[83]*

1. *Let Σ be a non-empty CNF that solved by the SLUR algorithm and $C \in \Sigma$ and a, b be two variables.*
   Let $\Sigma' = \Sigma \cup \{C \cup \{a, b\}, \{C \cup \{a, \neg b\}, \{C \cup \{\neg a, b\}, \{C \cup \{\neg a, \neg b\}\}$.
   The SLUR algorithm solves Σ' by property 23 but Σ' does not belong to the hidden extended Horn class by example 10.

2. *Let V be a set of three or more variables and Σ be a CNF emphasizing that an even number of variables of V are TRUE.*
   For example if V has exactly three variables, i.e.,
   $V = \{a, b, c\}$ then $\Sigma = \{\{\neg a, b, c\}, \{a, \neg b, c\}, \{a, b, \neg c\}, \{\neg a, \neg b, \neg c\}$.
   Σ is solved by SLUR but Σ does not belong to both hidden extended Horn class (by example 10) and balanced formulas (by checking its matrix).

Algorithm 9: SLUR algorithm $SLUR(\Sigma)$[82, 83]

```
Input: A CNF formula F with no empty clause ;
Output: A partial truth assignment satisfying F,UNSAT, or give-up;
(F, t) ← UP(F) ;
if F contains an empty clause then
    return UNSAT ;
while F is not empty do
    Select a variable x present in F;
    (F1, t1) ← UP(F ∧ x̄);
    (F2, t2) ← UP(F ∧ x);
    if both F1 and F2 contain an empty clause then
        return "give up";
    if F1 contains an empty clause then
        (F, t) ← (F2, t ∪ t2);
        else if F2 contains an empty clause then (F, t) ← (F1, t ∪ t1);
        else Choose one of the following two steps;
        (F, t) ← (F1, t ∪ t1);
        (F, t) ← (F2, t ∪ t2);
return t ;
```

The SLUR class is not believed to be recognizable in polynomial time (in fact checking whether a CNF belongs to SLUR is Co-NP-Complete [84]).
In Algorithm 9, $UP(F)$ returns (F', t) where

1. F' is the result of applying unit propagation on F, and

2. t is the set of unit clauses found by unit propagation.

Theorem 8. *[77]*
Let Σ be a CNF. The SLUR algorithm exhibits an $O(\Sigma)$ time complexity if both calls to UP are applied simultaneously and one call is immediately abandoned if the other finishes first without falsifying a clause.

Example 12. $\Sigma = \{\{x_1, x_2, \neg x_4\}, \{\neg x_1, \neg x_3, x_4\}, \{\neg x_2, x_3, \neg x_5\}\}$ *is SLUR formula .*

Some extensions and variants of the SLUR algorithm

In this section we indicate some possible extensions of the SLUR algorithm; see section 3.6.6 for other possible extensions.

Remark 14. *[77]*

1. *One can extend the SLUR algorithm by adding a 2SAT algorithm to the unit resolution steps of the SLUR algorithm.*
   This extended-SLUR algorithm can solve all 2SAT instances while SLUR algorithm cannot.

2. *Also, we can extend the SLUR algorithm by allowing a polynomial number of backtracks, by giving up if at least one branch of the search tree does not terminate at a leaf where a clause is falsified.*
   This variant of the SLUR algorithm can solve unsatisfiable CNFs that have short search trees.

3.3.4 Almost-Horn, $F-Horn^*$, ordered, ordered-renamable and almost ordered formulas

Almost-Horn class

The Almost-Horn class was introduced in[85], its definition class depends on the the so-called Horn basis notion.
First, we need the definitions of X-Horn CNFs and X-Horn- renamable CNFs. If Σ is a CNF and $X \subseteq V(\Sigma)$, then $L(X)$ denotes the literals from X.

Definition 114. *(X-Horn CNF)[85]*
Let Σ be a CNF and let $X \subseteq V(\Sigma)$. Σ is X-Horn if and only if every clause of Σ contains a positive literal from $L(X)$ is a Horn clause on X.

Definition 115. *(X-Horn-renamable CNF)[85] [86]*
Let Σ be a CNF and let $X \subseteq V(\Sigma)$. Σ is X-Horn-renamable if and only if it can be obtained from a X-Horn CNF by renaming some of its variables.

Notation 8. *[85]*
Let Σ be a CNF. $Rest(\Sigma, X) = \{C \in \Sigma : C \cap L(X) = \emptyset\}$.

Property 24. *[85]*
Let Σ be X-Horn-renamable without unit clauses. Σ is satisfiable if and only if $Rest(\Sigma, X)$ is satisfiable.

Property 25. *[85]*
Let Σ be a CNF and let $X_1 \subseteq V(\Sigma)$ and $X_2 \subseteq V(\Sigma)$. If Σ is X_1-Horn-renamable and X_2-Horn-renamable, then Σ is $X_1 \cup X_2$-Horn-renamable.

Definition 116. *(Horn basis)[85]*
Let $X_i \subseteq V(\Sigma)$ $(i = 1, 2, \ldots, k)$ be all the sets such that Σ is X_i-Horn-renamable and $B = (X_1 \cup X_2 \cup \cdots \cup X_k)$ then Σ is B-Horn-renamable. The set B is called the Horn basis of Σ.

Notation 9. *[85]*
Let Σ be a CNF. The Horn basis of Σ is denoted by $Basis(\Sigma)$ and $Rest(\Sigma) = Rest(\Sigma, Basis(\Sigma))$.

Property 26. *[85]*
Let Σ be a CNF without unit clauses. Σ is satisfiable if and only if $Rest(\Sigma)$ is satisfiable.

So, for a CNF Σ without unit clauses (note that we can always remove unit clauses from any CNF using UP), $Basis(\Sigma) \neq \emptyset \Rightarrow Rest(\Sigma) \subsetneq \Sigma$, and then $Basis(Rest(\Sigma)) \neq \emptyset \Rightarrow Rest(Rest(\Sigma)) \subsetneq Rest(\Sigma)$ and so on, hence we can repeat until we get a CNF with empty Horn basis.
So, we have the following definition.

Definition 117. *$Iterated - Rest(\Sigma)$[85]*
Let Σ be a CNF.

$$Iterated - Rest(\Sigma) = \begin{cases} \Sigma & if\ \ Basis(\Sigma) = \emptyset \\ Iterated - Rest(Rest(\Sigma)) & otherwise \end{cases}$$

Property 27. *[85]*
Let Σ be a CNF. Σ is satisfiable if and only if $Iterated - Rest(\Sigma)$ is satisfiable.
So, if $Iterated - Rest(\Sigma) = \emptyset$ then Σ is satisfiable.

Property 28. *(almost-Horn class)[85, 87]*
Σ is almost-Horn CNF if and only if $Iterated - Rest(\Sigma) = \emptyset$.

To find $Iterated - Rest(\Sigma)$, one has to compute $Basis(\Sigma)$, the authors in [85] have presented a linear algorithm ($O(|\Sigma|)$ complexity) to compute $Basis(\Sigma)$, this is done by using Aspvall's algorithm that reduces Horn-renaming problem to 2-SAT (see section 3.3.2), and solves the 2-SAT problem (see section 3.3.1).

After computing $Basis(\Sigma)$, one can compute $Rest(\Sigma)$ in $O(|\Sigma|)$ complexity and if $Basis(\Sigma) \neq \emptyset$ then $V(Rest(\Sigma)) \subset V(\Sigma)$, so compute $Iterated - Rest(\Sigma)$ needs at most n steps and so has $O(n|\Sigma|)$ complexity time.

$F - Horn^*$ class

Luquet in his thesis [86] extended the class almost-Horn to the class $F - Horn^*$ as follows.

Definition 118. *$F - Horn^*$ class [86]*
Let Σ be a CNF and let F be a tractable class of SAT, $\Sigma \in F - Horn^*$ class if and only if Σ has no unit clause and $Iterated - Rest(\Sigma) \in F$.

We have the following about the time complexity of recognition and solving algorithms for $F - Horn^*$ class.

Notes 15. *[86]*
Let Σ be a CNF with n variables and Let F be a tractable class of SAT such that the recognition and solving algorithms for F have $O(f(\Sigma))$ and $O(g(\Sigma))$ time complexity respectively. The recognition and solving algorithms for $F - Horn^*$ class have $O(n|\Sigma| + f(\Sigma))$ and $O(|\Sigma| + g(\Sigma))$ time complexity, respectively.

One can recognize q-Horn class (see definition 128) by using a $F - Horn^*$ class, more precisely.

Notes 16. *[86]*
q-Horn is the $(2SAT) - Horn^*$ class.

Ordered formulas

Unit propagation does not only solve the satisfiability problem for Horn class (see section 3.3.2), but it can also be used to generate efficiently the models of any Horn CNF. The authors in [87] generalize Horn class to classes (ordered formulas, ordered-renamable formulas and also almost ordered formulas but on condition that the variables are suitably ordered) that preserve these computational properties of unit propagation. The other classes that benefit from these properties of unit propagation are extended Horn formulas (see section 3.3.3), simple extended Horn (see section 3.3.3) and balanced formulas (see section 3).

Definition 119. *(free/linked literal)[87]*
Let $C \in \Sigma$ and $l \in C$. l is linked in C (with respect to Σ) if $Occ(\neg l) = \emptyset$ or $\exists t \in C$ $(t \neq l)$ such that $Occ(\neg l) \subseteq Occ(\neg t)$.$l$ is free in C if and only if it is not linked in it.

Definition 120. *(ordered formulas)[87]*
Σ is ordered if $\forall C \in \Sigma$, C has at most one positive literal free in it.

The authors in [87] gave a recognition algorithm that, for each literal, computes the set of literals linked to it and then counts the number of free positive literals of each clause. The time complexity of this algorithm is $O(n|\Sigma|)$. For solving algorithm, UP is sufficient for solve the satisfiability problem for ordered formulas:if the empty clause is not obtained using UP then the given CNF is satisfiable, so the time complexity of this algorithm is $O(|\Sigma|)$.

Theorem 9. *[87] Let n be the number of variables in Σ and $|\Sigma|$ be the number of occurrences of literals in Σ. Whether Σ is ordered or not can be decided in $O(n|\Sigma|)$ time and the satisfiability problem of Σ can be solved in $O(|\Sigma|)$ time.*

Example 13. *$\Sigma = \{\{x_1, x_2, x_3, x_4\}, \{\neg x_1, x_2, \neg x_3\}, \{\neg x_1, \neg x_2, \neg x_4, x_5\}, \{\neg x_1, x_3, x_4, \neg x_5\}\}$ is an ordered formula.*

Example 14. *[87]*

1. *Every Horn CNF is an ordered CNF.*
2. *If a CNF consists of positive literals then it is an ordered CNF.*
3. *$\Sigma = \{\{x_1, x_2\}, \{x_1, x_3\}, \{x_2, x_3\}\}$ is ordered but not extended Horn.*

Definition 121. *(Unit derivation, $Unit(\Sigma)$,$Kernel(\Sigma)$) [87]*

1. *(Unit derivation)*
   A unit derivation from a CNF Σ is a finite sequence $(C_1, \ldots, C_p)$ of clauses such that for every $i, 1 \leq i \leq p$ either $C_i \in \Sigma$ or $\exists j, k \in \{1, \ldots, i-1\}$ and a literal l such that $C_j = C_i \cup \{\neg l\}$ and $C_k = \{l\}$.
   A clause C is said to be derivable from Σ by unit resolution if and only if there exists a unit derivation from Σ,$(C_1, \ldots, C_p)$ such that $C_p = C$.
2. *$Unit(\Sigma) = \{l \in L(V) : \{l\}$ derives from Σ by unit resolution$\}$.*
3. *$Kernel(\Sigma) = \Sigma|_{Unit(\Sigma)}$.*

Property 29. *[87]*

Σ is satisfiable if and only if $Unit(\Sigma)$ is consistent and $Kernel(\Sigma)$ is satisfiable.

Remark 15. *[87]*
Σ is said to satisfy property P^* if for every set U of unit clauses on V, $\Sigma \cup U$ is satisfiable if and only if $Unit(\Sigma \cup U)$ is consistent.

Theorem 10. *[87]*
If a CNF Σ satisfies property P^* then the satisfiability problem for Σ can be solved in $O(\Sigma)$ time.

Property 30. *[87]*

1. *If every clause in Σ contains a negative literal or a literal linked in C then Σ is satisfiable.*
2. *If Σ is a Horn formula and contains no positive unit clause then Σ is satisfiable.*
3. *If Σ is ordered and contains no positive unit clause $C = \{x\}$ such that x is free in C then Σ is satisfiable.*
4. *Let U be a set of unit clauses on V. If Σ is ordered then $Kernel(\Sigma \cup U)$ is ordered.*
5. *If Σ is ordered then Σ satisfies P^*.*

Ordered-renamable formulas

The authors in [87] introduced also the ordered-renamable class which also satisfies property P^* and so the satisfiability problem for Σ can be solved in $O(\Sigma)$ time, too.

Definition 122. *(ordered-renamable formula)[87]*
Σ is ordered-renamable if it can be transformed into an ordered formula by renaming some variables.

Property 31. *[87]*

1. *If Σ' is obtained from a formula Σ by renaming some variables and Σ satisfies P^* then Σ' satisfies P^*.*

2. *The satisfiability problem for ordered-renamable CNF Σ can be solved in $O(|\Sigma|)$ time and the recognition algorithm for ordered-renamable CNF Σ can be solved in time $O(n|\Sigma|)$.*

Almost ordered formulas

The third class that was introduced in [87] is the so-called almost ordered class. It can be recognized in polynomial time but it does not satisfy Property P^*.
The definition of almost ordered formulas is based on the notion of ordered basis, which generalizes the Horn basis.

Definition 123. *(X-ordered formula, X-ordered-renamable formula)[87]*
Let X be a subset of variables of a CNF Σ. Σ is X-ordered if for every clause C in Σ and every positive literal x in X, if $x \in C$ and x is free in C then $C \in L(X)$ and x is the only positive literal free in C. Σ is X-ordered-renamable if and only if we can transform Σ into an X-ordered formula by renaming some variables.

X-ordered formula, X-ordered-renamable formula are generalizations of ordered formula and ordered-renamable formula, respectively.

Notes 17. *[87]*
Σ is ordered if and only if Σ is V-ordered whereas Σ is ordered-renamable if and only if Σ is V-ordered-renamable.

Property 32. *[87]*
Let Σ be a CNF.

1. *Let $X \subseteq V$ and let $Rest(\Sigma, X) = \{C \in \Sigma : C \cap L(X) = \emptyset\}$ and assume that Σ is X-ordered-renamable without unit clauses. We have that Σ is satisfiable if and only if $Rest(\Sigma, X)$ is satisfiable.*

2. *Let $X_1 \subseteq V$ and $X_2 \subseteq V$. If Σ is X_1-ordered-renamable and X_2-ordered-renamable then Σ is $(X_1 \cup X_2)$-ordered-renamable.*

Definition 124. *(ordered basis) [87]*
Let $X_1, \ldots, X_k$ be all the subsets of V such that Σ is X_i-ordered-renamable and $B = X_1 \cup \cdots \cup X_k$. From Property 32 Σ is B-ordered-renamable. The set B is called the ordered basis of Σ and denoted by $OBasis(\Sigma)$.

Property 33. *[87]*

1. *Every Horn basis (X-Horn formula, X-Horn-renamable formula) is ordered basis (X-ordered formula, X-ordered-renamable formula, respectively).*

2. *Let $ORest(\Sigma) = Rest(\Sigma, OBasis(\Sigma))$. If Σ contains no unit clause, then Σ is satisfiable if and only if $ORest(\Sigma)$ is satisfiable. So, if Σ contains no unit clause and $ORest(\Sigma) = \emptyset$ then Σ is satisfiable.*

3. *The ordered basis of Σ can be computed in $O(n|\Sigma|)$ time.*

Definition 125. *Iterated − ORest(Σ)[87]*
Let Σ be a CNF.
$$Iterated - ORest(\Sigma) = \begin{cases} \Sigma & if\ OBasis(\Sigma) = \emptyset \\ Iterated - ORest(ORest(\Sigma)) & otherwise \end{cases}$$

Definition 126. *(almost ordered formula)[87]*
Σ is almost ordered if and only if Iterated − ORest(Σ) = ∅.

Notes 18. *[87]*
If Σ is almost ordered and contains no unit clauses then Σ is satisfiable.

Definition 127. *(suitable permutation)[87]*
Let Σ be an almost ordered CNF. There exist $X_1, \ldots, X_k$, $X_i \subseteq V$ $(1 \leq i \leq k)$, and $\Sigma_1, \ldots, \Sigma_k$, $\Sigma_i \subseteq \Sigma$ $(1 \leq i \leq k)$, such that $\Sigma_1 = \Sigma, X_i = OBasis(\Sigma_i)$ $(1 \leq i \leq k)$, $\Sigma_{i+1} = ORest(\Sigma_i)$ $(1 \leq i \leq k-1)$ and $\Sigma_k = \emptyset$. Let $W = V \setminus (X_1 \cup \cdots \cup X_k)$. A permutation $(x_1, \ldots, x_n)$ of the variables of Σ is suitable if and only if for every j $(1 \leq j \leq n)$, $\{x_1, \ldots, x_j\} \subseteq W$ or there exists i such that $\{x_1, \ldots, x_j\} = W \cup X_k \cup X_{k-1} \cup \cdots \cup X_{i+1} \cup X$ with $X \subseteq X_i$.

Theorem 11. *[87]*
One can decide whether Σ is almost ordered, and build (if Σ is almost ordered) a suitable permutation, in $O(n^2|\Sigma|)$ time.

3.3.5 q-Horn, Matched, Generalized Matched and LinAut formulas

q-Horn formulas

Definition 128. *(q-Horn class)*
Let Σ be a CNF that has m clauses and n variables. Let $|C_i|$ be the length of the clause C_i (i.e., the number of literals occurring in it) and let w be the m-vector such that its components are $|C_i|$ and let x be a real n-vector and $\mathbf{1}$ be the vector of 1's (the dimension of this vector is recognized from the context). Consider the system of real inequalities
$M_\Sigma(x) \leq 2Z\mathbf{1} - w, \quad -\mathbf{1} \leq x \leq \mathbf{1}$.
The CNF is called q-Horn if and only if this system is satisfied for $Z = 1$. [79, 82, 88, 89].

Remark 16. *(special cases of q-Horn) [79, 82, 88, 89].*
If the inequalities of the system is satisfied by $Z = 1$ and all the components of x are 1 then the q-Horn CNF is Horn CNF, and if the inequalities of the system is satisfied by $Z = 1$ and all the components of x are ± 1 then the q-Horn CNF is renamable Horn CNF.

Theorem 12. *[79, 82, 88, 89].*
Let c be a fixed positive real number, the CNFs that satisfy $M_\Sigma(x) \leq 2Z\mathbf{1} - w$, $-\mathbf{1} \leq x \leq \mathbf{1}$ with $Z = 1 + \frac{clogn}{n}$ can be solved in polynomial time. Let β be a real number, $\beta < 1$, the CNFs that satisfy $M_\Sigma(x) \leq 2Z\mathbf{1} - w$, $-\mathbf{1} \leq x \leq \mathbf{1}$ with $Z = 1 + \frac{1}{n^\beta}$ is NP-complete.

Example 15. $\Sigma = \{\{x_4, \neg x_5, \neg x_6\}, \{\neg x_4, \neg x_5, \neg x_6\}, \{\neg x_6, x_7\}, \{\neg x_4, \neg x_5, x_1, x_2\}, \{\neg x_4, \neg x_6, \neg x_2, x_3\}, \{\neg x_5, \neg x_7, x_2, x_3\}, \{\neg x_2, \neg x_3\}, \{x_2, \neg x_3\}, \{x_1, x_2\}\}$ *is a q-Horn instance.*

Notes 19. *[89] Σ is q-Horn if and only if the variables of Σ can be decomposed uniquely (up to renaming of Σ) into two sets H and Q, $H \cap Q = \emptyset$ such that no clause of Σ has:*

1. *More than two variables of Q.*
2. *More than one positive literal of H.*
3. *One positive literal of H and a literal of Q.*

Notes 20. *In example 15, $Q = \{x_1, x_2, x_3\}$ and $H = \{x_4, x_5, x_6, x_7\}$.*

Notes 21. *[79, 82, 88, 89]*
For a CNF Σ, after determining the two sets H and Q that satisfy the above three conditions, which can be done in O|(Σ|), (and hence recognition that Σ belongs to q-Horn),the decision algorithm consists in assigning the FALSE value to variables of H and in solving the resulting 2SAT by any linear algorithm for 2SAT class, see section 3.3.1. Hence both the recognition algorithm and the decision algorithm for q-Horn class run in linear time.

Maximum monotone decomposition characterization of the q-Horn class

q-Horn class can be characterized by using maximum monotone decomposition of matrices [77]

Definition 129. *(Maximum monotone decomposition of matrices) [77]*
Let M_Σ be the (0,±1)-clause-variable matrix of the CNF Σ. In the monotone decomposition of M_Σ the columns are multiplied by -1 and the rows and columns are partitioned into submatrices $\begin{pmatrix} A_1 & E \\ D & A_2 \end{pmatrix}$ *where*

1. *Each row in the submatrix A_1 has at most one $+1$.*
2. *The submatrix D contains only -1 or 0 entries.*
3. *There is no restrictions on the submatrix A_2.*
4. *The submatrix E has 0 entries only.*

The decomposition is a maximum monotone decomposition if and only if A_1 is the largest possible over columns.

Property 34. *(q-Horn formula as maximum monotone decomposition)[77]*
The CNF represented by M_Σ is q-Horn if in the maximum monotone decomposition of M_Σ, A_2 has no more than two nonzero entries per row.

Property 35. *[77, 90]*
One can find in linear time a maximum monotone decomposition for a matrix associated with a q-Horn formula, and after putting q-Horn formula in this form, it can be solved in linear time.

We can use a similar matrix decomposition for the polynomial class in Theorem 12

Remark 17. *[77]*
The M_Σ matrix for the polynomial class in Theorem 12, after multiplying by -1, can be partitioned as $\begin{pmatrix} A_1 & E & B_1 \\ D & A_2 & B_2 \end{pmatrix}$ *where A_1, E, D, A_2 as for q-Horn CNFs and the number of columns of B_1 and B_2 is less than or equal $O(ln(n))$.*

Also we get the following.

Property 36. *[77]*
The satisfiability problem for the CNFs in Theorem 12 can be solved in polynomial time by solving the q-Horn system that is obtained by substituting each of $2^{O(ln(n))}$ partial truth assignments to the variables of B_1 and B_2.

Matched formulas

Definition 130. *(Matched formulas)[82]*
Let Σ be a CNF and $UBG(\Sigma)$ is the undirected bipartite graph with vertex sets $C(\Sigma)$ and $V(\Sigma)$ where $(a, C) \in UBG(\Sigma)$ if and only if $a \in C$ or $\neg a \in C$ that is $M_\Sigma(i, j) \neq 0$.
A total matching for the set $C(\Sigma)$ is a set $M \subset EDG(UBG(\Sigma))$ such that if $C \in C(\Sigma)$,C is incident upon exactly one edge of M and if $a \in V(\Sigma)$, a is incident upon at most one edge of M.
If $UBG(\Sigma)$ has a total matching then Σ is called a matched CNF.

Finding a total matching for an undirected bipartite graph (actually for any undirected graph) can be done in polynomial time. A total matching for $UBG(\Sigma)$ (if one exists) can be found by using an augmenting path algorithm.
The following lemma shows that if a total matching for $UBG(\Sigma)$ exists then Σ is satisfiable. Because if M is a total matching for $UBG(\Sigma)$ then the edges of this matching give a satisfying partial assignment for Σ ;$\forall edge(v, C) \in M$, if $\neg v \in C$ then assign the value $FALSE$ to v else assign the value $TRUE$ [82].

Property 37. *[82]. Let Σ be a CNF. If there is a total matching for $UBG(\Sigma)$ then Σ is satisfiable.*

Hence, a recognition algorithm for matched formulas Σ can run in polynomial time, by checking if $UBG(\Sigma)$ has a total matching. The same algorithm is a decision algorithm for this class because if the total matching exists for $UBG(\Sigma)$ then Σ is satisfiable.

Example 16. $\Sigma = \{\{x_1, x_2, \neg x_4\}, \{\neg x_1, \neg x_3, x_4\}, \{\neg x_2, x_3, \neg x_5\}\}$ *is a matched formula.*

Generalized matched formulas

Szeider [91] generalized the class of matched formulas by the so-called biclique subgraph.

Definition 131. *(Biclique) [77, 91]*
Let Σ be a CNF and $UBG(\Sigma)$ is the undirected bipartite graph with vertex sets $C(\Sigma)$ and $V(\Sigma)$ where $(a, C) \in UBG(\Sigma)$ if and only if $a \in C$ or $\neg a \in C$ and let $C_1 \subseteq C(\Sigma)$ and $V_1 \subseteq V(\Sigma)$. A biclique is the complete (i.e. for each $v \in V_1$ and $C \in C_1$ there is $e = (v, C) \in E$) subgraph $G(C_1, V_1, E)$ of $UBG(\Sigma)$ induced by C_1 and V_1.

If all the bicliques of $UBG(\Sigma)$ are single edges then Σ is a matched formula.

Theorem 13. *[91]*
Let Σ be a CNF with n variables and m clauses. If $UBG(\Sigma)$ is a biclique and $m < 2^n$ then Σ is satisfiable.

Example 17. *[91]*
The CNF $\Sigma = \{\{x_1, x_2\}, \{x_1, \neg x_2\}, \{\neg x_1, x_2\}\}$ is satisfiable according to Theorem 13 but it is not matched.

We can check the satisfiability of a CNF by covering all clauses of $UBG(\Sigma)$ by several bicliques under the condition that every variable belongs to at most one biclique.

Theorem 14. *[91]*
If there is a collection $C = \{X_1, \ldots, X_k\}$ of sets of vertices of $UBG(\Sigma)$ such that

1. *Every clause of Σ belongs to some X_i, $1 \leq i \leq k$.*
2. *Every variable of Σ belongs to at most one X_i, $1 \leq i \leq k$.*
3. *X_i induces in $UBG(\Sigma)$ a biclique on n_i variables and $m_i < 2^{n_i}$ clauses, $1 \leq i \leq k$.*

then Σ is satisfiable.

Definition 132. *(biclique satisfiable(generalized matched formulas)) [77, 91]*
A CNF formula is biclique satisfiable (generalized matched formulas) if and only if it satisfies the conditions in theorem 14.

Unfortunately we have the following result for recognizing a generalized matched formula.

Notes 22. *[77]*
The problem of recognition a generalized matched formula is NP-complete, even for 3-CNF formulas.

We can choose the collection $C = \{X_1, \ldots, X_k\}$ in Theorem 14 such that all elements of C are mutually disjoint.

Corollary 3. *[91]*
If there is a mutually disjoint collection $C = \{X_1, \ldots, X_k\}$ of sets of vertices of $UBG(\Sigma)$ such that

1. *Every clause of Σ belongs to some X_i, $1 \leq i \leq k$.*
2. *Every variable of Σ belongs to at most one X_i, $1 \leq i \leq k$.*
3. *X_i induces in $UBG(\Sigma)$ a biclique on n_i variables and $m_i < 2^{n_i}$ clauses, $1 \leq i \leq k$.*

then Σ is satisfiable.

If we restrict the size of bicliques by some positive integer $l \geq 1$ then we get the following special case of biclique satisfiable.

Definition 133. *(l-biclique satisfiable) [91]*
A CNF Σ is l-biclique satisfiable if and only if there is a collection $C = \{X_1, \ldots, X_k\}$ of sets of vertices of $UBG(\Sigma)$ such that

1. *Every clause of Σ belongs to some X_i, $1 \leq i \leq k$.*
2. *Every variable of Σ belongs to at most one X_i, $1 \leq i \leq k$.*
3. *X_i induces in $UBG(\Sigma)$ a biclique on n_i variables and $m_i < 2^{n_i}$ clauses, $1 \leq i \leq k$.*
4. *Each X_i contains at most l variables.*

The recognition algorithm of 1-biclique satisfiable has polynomial time complexity but for all l,$l \geq 2$ it is NP-complete.

Property 38. *(The recognition algorithm of l-biclique satisfiable) [91]*

1. *A CNF formula is matched if and only if it is 1-biclique satisfiable. Hence 1-biclique satisfiable CNF formulas can be recognized in polynomial time.*
2. *For any $l \geq 2$, recognition of l-biclique satisfiable 3- CNF formulas is NP-complete.*

Both matched formulas and biclique satisfiable (generalized matched formulas) are special cases of Var-satisfiable CNF formulas.

Definition 134. *(Var-satisfiable formulas)[91]*
A CNF formula Σ is var-satisfiable if and only if every CNF formula Σ' such that $UBG(\Sigma') = UBG(\Sigma)$ is satisfiable.

Every generalized matched formula (and hence every matched formula) is var-satisfiable formula but the converse is not true.

Property 39. *[91]*

1. *A CNF is var-satisfiable if and only if it is satisfiable and remains satisfiable under arbitrary replacement of literals by their complements.*
2. *Since polarity of literals is not used to form the incidence graphs, every matched formula and every generalized matched formula is var-satisfiable formula.*
3. *There are var-satisfiable CNF formulas that are not biclique satisfiable: $\Sigma = \{\{x_1, x_2, x_3\}, \{\neg x_1, x_2, x_3\}, \{\neg x_1, x_2, \neg x_3\}, \{x_1, \neg x_2\}, \{\neg x_1, \neg x_2\}\}$ is a var-satisfiable CNF but is not biclique satisfiable.*

LinAut formulas

Kullmann [92] introduced the Linear Autarky and defined the LinAut class. To present the LinAut class, we need the autarky definition

Definition 135. *(autarky)[92, 25, 93]*
Let τ be a partial assignment (represented by the set of literals that it satisfied, see remark 2) and Σ be a CNF (represented by a family of sets where its clauses are represented as sets of their literals).
τ is an autarky for Σ if and only if $\forall C \in \Sigma, V(\tau) \cap V(C) \neq \emptyset \Rightarrow \tau(C) = TRUE$.

Example 18. *[94]*
A pure literal is an autarky.

As a special case we have the linear autarky.

Definition 136. *(linear autarky)[92, 94]*
A linear autarky is a real n-vector x, $x \neq \mathbf{0}$ which satisfies the following system of inequalities.
$M_\Sigma(x) \geq \mathbf{0}$.

Example 19. *[94]*
A pure literal is a linear autarky.

Property 40. *[92, 94]*

1. *Every linear autarky induces an autarky.*
2. *We can decompose (we call this decomposition DEC) a given CNF into two parts using the autarky induced by the linear autarky.*
   (a) *The first part is satisfied by the autarky.*
   (b) *The second part contains just the variables that undefined in the autarky.*
   (c) *The second part is satisfiable if and only if the given CNF is satisfiable.*
   (d) *If we recursively use the decomposition DEC then we finally get a CNF that is semantically equivalent to the given CNF and this resulting CNF has no linear autarky (i.e., linear autarky-free subformula of the given CNF formula).*
   (e) *We can use linear programming to find the linear autarkies in polynomial time.*
   (f) *The satisfiable 2SAT is solved by using the decomposition DEC; this means two things:*
      i. *A satisfying truth assignment is an autarky induced by a linear autarky.*
      ii. *If the the resulting linear autarky-free subformula is non empty then the given 2SAT is unsatisfiable.*

We now define the LinAut class.
First, we define the class *LinAut* of CNFs with at least three literals per clause [82]
(LinAut of CNFs with at least three literals per clause)
Let us denote the LinAut class of CNFs with at least three literals per clause by $LinAut_{\geq 3}$. This definition is easier than the general definition since we do not need to deal with the 2SAT subformulas.
Note that a satisfiable CNF belongs to $LinAut_{\geq 3}$ if and only if we get the empty CNF using the decomposition *DEC*.

Definition 137. *($LinAut_{\geq 3}$ formulas)[82]*
Let Σ be a CNF with at least three literals per clause. The $LinAut_{\geq 3}$ class is defined inductively as follows.

1. *The empty CNF is in $LinAut_{\geq 3}$.*

2. *Suppose Σ has a linear autarky x.*
   $linSat(\Sigma, x) = \{C : C$ is a clause that has a literal v_j (or $\neg v_j$) $\in C$ that corresponds to $x_j \in x, x_j \neq 0\}$.
   let $\Sigma' = \Sigma \setminus (linSat(\Sigma, x))$, if $\Sigma' \in LinAut_{\geq 3}$ then $\Sigma \in LinAut_{\geq 3}$.
   We can construct a partial assignment τ that satisfies $linSat(\Sigma, x)$ without affecting Σ', i.e., Σ is satisfiable if and only if Σ' is satisfiable. The partial assignment τ is constructed as follows.

   (a) *Let v_j take the value TRUE if $x_j > 0$ and the value FALSE if $x_j < 0$ (if the clause $C_i \in linSat(\Sigma, x)$ then at least one of its literals takes the value TRUE since $M_\Sigma(i, j)(x_j)$ takes at least one positive value form all the nonzero values).*

   (b) *If $C_i \in \Sigma'$ then we do not assign a value by τ to all literals of C_i.*

(The general definition of LinAut formulas)
Here we need to take into account the 2SAT subformulas of the given CNF formula, but since the satisfiable 2SAT is completely solved by using the decomposition *DEC*, we can define the general LinAut class as follows.

Definition 138. *(LinAut class) [94]*
A CNF belongs to LinAut class if and only if by using the decomposition DEC

1. *either we get the empty set,*

2. *or we get a linear autarky-free 2SAT subformula.*

Notes 23. *[94]*
In definition 138 a CNF belongs to LinAut class is

1. *satisfiable if we get the empty set by using the the decomposition DEC*

2. *unsatisfiable if we get a linear autarky-free 2SAT subformula by using the decomposition DEC.*

By using the complexity index, some properties can be derived on linear autarky-free formulas and on the relationship between q-Horn class (see definition 128) and LinAut class.

Definition 139. *(complexity index) [94]*
The complexity index of a CNF with n variables is the solution to the linear system
$$\begin{cases} \min Z \\ M_\Sigma(x) \geq L - 2Z\mathbf{1} \\ x \in [-1, 1]^n \end{cases}$$
where L is a vector that has the length of clause i in its i-th entry.

Property 41. *[94]*

- *q-Horn CNF without unit-clause is in the class LinAut.*

- *In a linear autarky-free formulas with complexity index Z we have;*

  1. *either for some clause i (with length L_i), $Z > \frac{1}{2}L_i$,*
  2. *or all clauses have the same length $L = 2Z$.*

- *A kSAT linear autarky-free formula has complexity index $Z = \frac{1}{2}k$.*

- *Let k be the length of the shortest clause of a CNF Σ with complexity index $Z < \frac{1}{2}k$: Σ is satisfiable.*

Example 20. $\Sigma = \{\{x_1, x_2, x_3\}, \{\neg x_1, \neg x_2, \neg x_3\}\}$ *is a LinAut formula.*

3.3.6 PURL algorithm

The authors in [95] have presented an algorithm (called PURL) that could solve the unsatisfiable instances that belong to SLUR or LinAut classes.

Definition 140. *$T_1(C, l)$ [96, 95]*
Let Σ be a CNF and $C \in \Sigma$ be a clause and $l \in C$.
$T_1(C, l)$ is the set of the truth assignments such that if $\tau \in T_1(C, l)$, $\tau(l) = TRUE$ and $\tau(t) = FALSE$ $\forall t (\neq l) \in C$.

Theorem 15. *[96] Let Σ be a satisfiable CNF. There is a clause $C \in \Sigma$ and a literal $l \in C$ and a truth assignment $\tau \in T_1(C, l)$ such that τ satisfies Σ, i.e., $\tau(\Sigma) = 1$.*

.

Definition 141. *(Removable literal) [95].*
A literal $l \in C$ is removable if and only if $\tau(\Sigma) = 0$ $\forall \tau \in T_1(C, l)$.

Theorem 16. *[95]. If Σ is a CNF and $C \in \Sigma$ is a clause and $l \in C$ is a removable literal then $(\Sigma \setminus C) \cup (C \setminus \{l\})$ and Σ are logically equivalent $(\Sigma \setminus C) \cup (C \setminus \{l\})$ is satisfiable if and only if Σ is satisfiable).*

Let $Flip(C, l) = (C \setminus \{l\}) \cup \{\neg l\}$ and $\overline{Flip(C, l)} = \{\neg l : l \in Flip(C, l)\}$ and let $X = \{l_1, \ldots, l_k\}$ be a consistent subset of literals of Σ, and let $\Sigma \cup U(X)$ denote the set $\Sigma \cup \{l_1\} \cup \ldots, \cup \{l_k\}$.

Property 42. *[95] If Σ is a CNF and $C \in \Sigma$ is a clause then $l \in C$ is a removable literal if and only if $\Sigma \vDash Flip(C, l)$.*

Definition 142. *(p-removable literal)*
Let Σ be a CNF and $C \in \Sigma$ be a clause and $l \in C$. l is p-removable if and only if $UP(\Sigma \cup U(\overline{Flip(C, l)})) = \emptyset$, i.e., $\Sigma \vDash^* Flip(C, l)$.

Since $\Sigma \vDash^* Flip(C, l)$ implies $\Sigma \vDash Flip(C, l)$, so if l is p-removable then l is removable and one can use the logically equivalence between $(\Sigma \setminus C) \cup (C \setminus \{l\})$ and Σ to remove all the p-removable literals from Σ and get a CNF without any p-removable literals (or reach the null clause).
The authors in [95] present this algorithm (called PURL). The running time of PURL belongs to $O(|\Sigma|^3)$.

The authors in [95] show that if an unsatisfiable CNF Σ belongs to the SLUR or LinAut classes (and hence to any of the classes (2SAT, Horn,reverse-Horn, CC-balanced,renamable Horn, extended Horn, q-Horn, Matched ones) then Σ can be solved using the PURL algorithm.

3.4 Some other satisfiable and polynomially solvable subclasses of SAT

In this section we introduce briefly some other satisfiable and polynomially solvable subclasses of SAT

3.4.1 Nested, extended nested formulas

In this section we present two tractable classes (nested and extended nested) that do not depend on the UP-technique.

Nested formulas

Let X be a linear ordered (by the ordered relation $<$, see definition 93) finite alphabet.
We associate the elements of X with the variables and literals of a CNF Σ as follows.

1. $v \in X$ if and only if $v \in V(\Sigma)$.

2. The positive literals in Σ are the elements of X and the negative literals in Σ are the others ones.

Notes 24. *[97].*

1. *The linear ordering of X can be extended to a linear preordering of all its literals by disregarding the signs.*

2. *A clause over X is a set of literals on distinct variables.*

Definition 143. *(the straddles of two clauses)[97]*
A clause C straddles clause C' if and only if there are literals $a, b \in C$ and $c \in C'$ such that $a < c < b$.

Definition 144. *(the overlap of two clauses)[97]*
Two clauses overlap if they straddle each other.

Definition 145. *(nested formulas)[97]*
A set of clauses in which no two overlap is called nested.

Nested formulas can be solved in linear time, basically this algorithm replace the clauses of the given CNF by a clauses of length two and check this replacement by a special form of dynamic 2SAT[97].

Example 21. $\Sigma = \{\{x_1, x_2\}, \{\neg x_1, x_4\}, \{\neg x_1, \neg x_4\}, \{x_1, \neg x_2, x_3\}\}$ *is nested formula.*

Extended nested formulas

Knuth [97] noted that the 2SAT formula arising from his algorithm, which works by replacing each clause by 2-length clause, is not completely general. So he suggested that there is larger tractable subclass of SAT that contains nested class, the authors in [98] found this larger class that extends the nested one. The authors in [98] defined the class of extended nested satisfiability problems by associating two hypergraphs to the CNF Σ.

1. The first hypergraph $H = (V(H), E(H))$ is as follows:

   - The vertices $V(H) = \{v_1, v_2, \ldots, v_n\}$ of H correspond to the variables $\{x_1, x_2, \ldots, x_n\}$ of Σ.
   - Each clause C_i of Σ corresponds to an edge $E_i \in E(H)$ of H where the vertices of $E_i \in E(H)$ correspond to the variables of C_i.

   Definition 146. *(large edge) [98]*
   An edge E is large if and only if it has more than two vertices.

2. The second hypergraph $\hat{H}$ is as follows.

   - (a) Let $W = \{v \in V(H) : \exists$ a clause $C, |C| \leq 2, x \in V(C)$, where x corresponds to v, and there is no clause $C', |C'| \geq 3, x \in V(C')\}$;
     (b) Let $H_W = (W, E_W)$ be the subhypergraph induced by W.
   - Let $W_i, i = 1, \ldots . \overline{w}$ be the connected components of H_W and N_i be its set of neighbors [i.e., the vertices $v_k \in V \setminus W$ and $v_l \in W_i$ such that $\{v_k, v_l\} \in H$].
     Then $\hat{H} = (V(\hat{H}), E(\hat{H}))$ is constructed from H as follows.
     (a) By setting $V(\hat{H})$ to $V(H)$ and $E(\hat{H})$ to $E(H)$.
     (b) By deleting from $E(\hat{H})$ all unary or binary edges contained in a large edge.
     (c) By adding to $E(\hat{H})$ large edges $E_i = (W_i \cup N_i)$ for each connected component W_i of H_W.

Definition 147. *(erasure operation on the hypergraph $\hat{H}$)[98]*
The erasure operation on the hypergraph $\hat{H}$ is done by either

1. *finding an edge E of $\hat{H}$ with at most one vertex belonging also to other edges,*

2. *removing that edge from $\hat{H}$.*

or

1. *finding a large edge E of $\hat{H}$ with exactly two vertices,v_k and v_l, which belong also to other edges,*
2. *removing that edge E from $\hat{H}$,*
3. *adding the edge $\{v_k, v_l\}$ to $\hat{H}$ if no edge of $\hat{H}$ contains both v_k and v_l.*

Definition 148. *(erasable hypergraph $\hat{H}$) [98]*
The hypergraph $\hat{H}$ is erasable if and only if the recursive application of erasure operation leads to remove all large edges.

Definition 149. *(extended nested class) [98]*
A CNF Σ defines an extended nested satisfiability problem if and only if the corresponding hypergraph $\hat{H}$ is erasable.

Then, the authors in [98] have introduced a linear time algorithm for solving and recognizing extended nested satisfiability instances: it consists in constructing the hypergraphs H and $\hat{H}$ for the given CNF Σ and in checking if $\hat{H}$ is empty or has no large edges; if the answer is positive then the algorithm solves the satisfiability for the 2SAT instance. Otherwise, it goes on by using an erasure operation to $\hat{H}$ to check if it is erasable or not. If $\hat{H}$ is erasable and the chosen edge is E then

1. Let V be the set of corresponding variables to its vertices.
2. Let Σ_v be the CNF that results from Σ by deleting all clauses that contain variables not in V.
3. Let T be the set of vertices of E that belong to other edges of $\hat{H}$.

We have three cases of T: T is empty or T has one vertex or T has two vertices, in all three cases the satisfiability of Σ depends on the satisfiability of subproblems, and in the case where Σ is satisfiable, a model of it is constructed from the models of the subproblems (see algorithm ENSAT in [98] for more details).

Example 22. *[98]*
$\Sigma = \{\{x_1, \neg x_2\}, \{x_2, x_6\}, \{x_1, \neg x_7\}, \{\neg x_4, \neg x_6\}, \{\neg x_6, x_{11}\}, \{\neg x_7, x_{10}\}, \{x_{10}, \neg x_{11}\}, \{x_7, \neg x_9\},$
$\{x_9, \neg x_{10}\}, \{x_2, \neg x_3\}, \{x_7, \neg x_8\}, \{x_3, x_5\}, \{\neg x_5, x_8\}, \{\neg x_1, \neg x_2, x_4, x_6, \neg x_7\}, \{x_6, x_7, x_9, x_{10}, x_{11}\}\}$ *is an extended nested CNF.*

Property 43. *[98]*

1. *Algorithm ENSAT recognizes and solves constructively extended nested satisfiability problems in linear time.*
2. *Nested satisfiability is a particular case of extended nested satisfiability.*

3.4.2 Affine formulas

Definition 150. *The affine class is a subclass of SAT that can be written as a system of linear equations where the coefficients and the solutions have to belong to* $\{0, 1\}$.

Example 23. *(For the symbol $\oplus$ see remark 1).*
Let $\{x_1, x_2, x_3, x_4, x_5, x_6\}$ be boolean variables. The following example is a system of affine formulas
$x_1 \oplus x_2 \oplus x_4 = 0$
$x_2 \oplus x_5 = 1$
$x_3 \oplus x_6 = 0$

The instances of affine class can be solved in polynomial time [61]. For example a satisfying assignment can be constructed for an affine instance by a variant of Gaussian elimination [99].

Notes 25.
1. *The solutions of the equation* $x_1 \oplus x_2 \oplus x_4 = 0$ *are exactly the same solutions of SAT instance* $(\neg x_1 \vee \neg x_2 \vee \neg x_4) \wedge (x_1 \vee x_2 \vee \neg x_4) \wedge (x_1 \vee \neg x_2 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$.
2. *The solutions of the equation* $x_2 \oplus x_5 = 1$ *are exactly the same solutions of SAT instance* $(\neg x_2 \vee \neg x_5) \wedge (x_2 \vee x_5)$.
3. *The solutions of the equation* $x_3 \oplus x_6 = 0$ *are exactly the same solutions of SAT instance* $(\neg x_3 \vee x_6) \wedge (x_3 \vee \neg x_6)$.
4. *So the solutions of the system of equations in example 23 are exactly the same solutions of SAT instance* $(\neg x_1 \vee \neg x_2 \vee \neg x_4) \wedge (x_1 \vee x_2 \vee \neg x_4) \wedge (x_1 \vee \neg x_2 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_4) \wedge (\neg x_2 \vee \neg x_5) \wedge (x_2 \vee x_5) \wedge (\neg x_3 \vee x_6) \wedge (x_3 \vee \neg x_6)$.

3.4.3 Doubly balanced 3SAT formulas and its extension to SAT formulas using balanced polynomial representation

The definition of doubly balanced 3SAT formulas [100] is based on the signs (negative or positive) of occurrences of the variables of the formula.

Definition 151. *(doubly balanced 3SAT formulas) [100]*
Let Σ *be 3SAT CNF with n variables and m clauses and let* M_Σ *be its clause-variable matrix and denote its* (i, j) *elements by* $a_{i,j}$.
Σ *is doubly balanced 3SAT formulas if and only if*

1. *for each variable* p_j, *the number of negative occurrences of this variable equals the number of positive occurrences of it, i.e.*
   $\sum_{i=1}^{m} a_{i,j} = 0$ *for all* $j = 1, \ldots, n$.
2. *For any two variables* p_j *and* p_k, *the number of clauses where both appear simultaneously with the same sign (i.e., both negative or both positive) is equal to the number of clauses in which both appear simultaneously with opposite signs (i.e., one appears negatively and the other positively), i.e.,*
   $\sum_{i=1}^{m} a_{i,j}a_{i,k} = 0$ *for all* $j, k = 1, \ldots, n,\ j \neq k$.

Property 44. *(doubly balanced 3SAT class is tractable) [100]*
A doubly balanced formula can be solved in polynomial time.

The authors in [100] generalized the notion of doubly balanced 3SAT class to any instances of SAT using a polynomial representation.

Definition 152. *(balanced polynomial representation and positive polynomial representation) [100]*
Let $c_I = (-1)^{|I|} \sum_{i=1}^{m} \prod_{j \in I} a_{i,j}$ *where* $I \subseteq \{1, \ldots, n\}$ *and let* (PR) *denote the problem.*

$$\text{Find } x = (x_1, \ldots x_n) \in \{-1, 1\}^n \text{ such that } P(x) = m + \sum_{I \subseteq \{1,\ldots,n\}} c_I \prod_{i \in I} x_i = 0. \qquad \ldots(PR)$$

1. *The polynomial function* $P(x)$ *is called balanced if* $\sum_{I \subseteq \{1,\ldots,n\}} |c_I| = m$.
2. *The polynomial function* $P(x)$ *is called positive if* $\sum_{I \subseteq \{1,\ldots,n\}} |c_I| < m$.

Given the polynomial representation (PR), we get the following [100].

Property 45. *Given a CNF Σ and its polynomial representation (PR).*

1. *If Σ has a positive polynomial representation then it is unsatisfiable.*
2. *If Σ has a balanced polynomial representation then it can be solved in $O(mn.min\{m,n\})$ time.*
3. *If $\sum_{I\subseteq\{1,\ldots,n\}} |c_I| = m + 2z$, then Σ can be solved in $O(\binom{m+2z}{z}mn.min\{m,n\})$ time.*

3.4.4 Exact linear and exact linearly-based formulas

A CNF formula is linear [101] if each pair of clauses have at most one shared variable; the authors in [101] proved that linear SAT and linear kSAT are NP-complete but there is a satisfiable subclass (exact linear formulas)[101] and polynomial subclass (exact linearly-based formulas) [102] of linear SAT.

Definition 153. *[101, 102]*
A CNF Σ without a complementary pair of unit clauses is

1. *linear if and only if for all $C_1, C_2 \in \Sigma$, $C_1 \neq C_2$ we have $|V(C_1) \cap V(C_2)| \leq 1$;*
2. *exact linear if and only if for all $C_1, C_2 \in \Sigma$, $C_1 \neq C_2$ we have $|V(C_1) \cap V(C_2)| = 1$;*
3. *exact linearly-based if and only if for all $C_1, C_2 \in \Sigma$, we have $|V(C_1) \cap V(C_2)| = 1$ or $V(C_1) = V(C_2)$.*

Property 46. *[101, 102]*
The satisfiability problem of

1. *linear formulas is NP-complete;*
2. *exact linear formulas is always satisfiable;*
3. *exact linearly-based formulas is polynomial time.*

Property 47. *[103]*
Let $m(k)$ be the largest integer m such that any linear k-CNF formula with less than or equal m clauses is satisfiable then $\frac{4^k}{4e^2k^3} \leq m(k) \leq ln(2)k^4 4^k$.

3.4.5 Formulas with many clauses

The author in [104] proved three results about CNFs with many clauses, using the following notation.

Notation 10. *[104]*
Let Σ be k-SAT with m clauses and n variables where each clause contains k different variables and $n \geq k \geq 1$. Let $k-SAT(> m_0)$ denote the $k-SAT$ problem of instances with $m > m_0(n)$ clauses.

Property 48. *[104]*

1. *Every instance of the $k-SAT(> \binom{n}{k}(2^k - 1))$ problem is unsatisfiable and this can be decided in polynomial time.*
2. *Every instance of the $k-SAT(> \binom{n}{k}(2^k - 1 - \frac{k}{n}))$ problem has at most one satisfying truth assignment and this can be decided in polynomial time and a satisfying truth assignment can be determined in polynomial time provided that there exists one.*
3. *For each $k \geq 3$ and each integer $l \geq 4$, the $k-SAT(> \binom{n}{k}(2^k - 1 - \frac{4}{l}))$ problem is NP-complete for $n \geq lk^2$.*

3.5 Minimally unsatisfiable formulas and maximum deficiency

An unsatisfiable CNF is minimally unsatisfiable if the result of removing any clause is a satisfiable CNF. To show how to solve and recognize a minimally unsatisfiable CNF, the definitions of deficiency and maximum deficiency are required.

Definition 154. *(deficiency of a CNF) [105, 77]*
The deficiency of a CNF is the difference between the number of its clauses and the number of its variables.

Every subformula of a CNF has its own deficiency which is the difference between the number of its clauses and the number of its variables. So we have the following definition.

Definition 155. *(maximum deficiency of a CNF) [105, 77]*
The maximum deficiency of a CNF is the maximum of the deficiencies of its subformulas.

Lemma 1. *(maximum deficiency of a minimally unsatisfiable CNF) [105, 77]*

1. *If Σ is a minimally unsatisfiable CNF with n variables then the number of clauses of Σ is at least $n+1$.*
2. *The maximum deficiency of a minimally unsatisfiable CNF is its deficiency that is the difference between the number of its clauses and the number of its variables.*

The maximum deficiency of a formula can be determined in polynomial time and the satisfiability of a CNF with a fixed maximum deficiency can be determined in polynomial time.

Theorem 17. *[105, 77]*

1. *Let Σ be a CNF formula with n variables and maximum deficiency δ. The satisfiability of Σ can be determined in $O(2^{\delta}n^3)$ time.*
2. *Let Σ be a minimally unsatisfiable CNF with n variables and deficiency δ. Then Σ can be recognized as a minimally unsatisfiable CNF in $O(2^{\delta}n^4)$ time.*

3.6 Hierarchies of Tractable Classes

In this section we focus on the hierarchies of tractable classes. The hierarchies of tractable classes consist of strata such that each stratum contains a tractable subclass of SAT and the classes in the hierarchy is defined recursively where the definition of the class in higher stratum depends on the definition of the class in lower stratum. There are many hierarchies of tractable classes in the literature, for example:

1. the hierarchy of Gallo and Scutella [1],
2. the hierarchy of Dalal and Etherington [2],
3. the two hierarchies of Cepek and Kucera [4],
4. the hierarchy of Pretolani [3],
5. the hierarchy of Kullmann [5],
6. the hierarchies that generalize SLUR class, $SLUR(i)$ [84],$SLUR^*(i)$ [106] and $SLUR_i$ [107, 108].
7. the hierarchy of Andrei et al. [109].

In this section we present some of these hierarchies briefly.

3.6.1 The hierarchy of Gallo and Scutella [1]

Yamasaki and Doshita [110] generalized the Horn class into a class that they called S_0, as follows

Definition 156. *A CNF $\Sigma = \{C_1, C_2, \ldots, C_n\} \in S_0$ if and only if*

1. *$C_i = P_i \cup H_i$, $i = 1, 2, \ldots, n$ where H_i is a Horn clause and P_i is a set of positive literals.*
2. *$P_i \subseteq P_{i+1}$, $i = 1, 2, \ldots, n-1$.*

We will call S_0 as S_0 or Γ_1 or Generalized Horn or GHorn.
They have presented an $O(|\Sigma|^2)$ algorithm for recognition S_0 and $O(|\Sigma|^3)$ algorithm to check the satisfiability of a CNF in S_0 (Arvind and Biswas [111] improved this result to $O(|\Sigma|^2)$. Note that Horn$\subseteq S_0$.

Gallo and Scutella [1] have generalized the Horn and S_0 classes and presented the following hierarchy .

Definition 157. *[1, 112].*
Let $\Gamma_0 :=$ HORN, a CNF $\Sigma = \{C_1, C_2, \ldots, C_n\} \in \Gamma_k$, where $k > 0$ if and only if

1. *$C_i = P_i \cup H_i$, $i = 1, 2, \ldots, n$ for some H_i where P_i is a set of positive literals.*
2. *$P_i \subseteq P_{i+1}$, $i = 1, 2, \ldots, n-1$.*
3. *$\{C_1 \setminus P_1, C_2 \setminus P_2, \ldots, C_n \setminus P_n\} \in \Gamma_{k-1}$.*

Note that $\Gamma_1 = S_0$.
Gallo and Scutella have presented two algorithms: one that recognizes instances in Γ_k in $O((|\Sigma| \times n^k)$ time and one for solving the instances in Γ_k in $O((|\Sigma| \times n^k)$ time, too.
Here, we introduce the hierarchy of Gallo and Scutella and adopt the same terminologies and definitions that these authors used in their paper (see [112] for equivalent definitions and terminologies).

Definition 158. *[1]*
Let $S = \{X_1, X_2, \ldots, X_n\}$ be a family of sets X_i such that $X_i \subseteq I$ where I is a set with $|I| = n$.

1. *Define the set S_J as $S_J = S \setminus \{X \in S : J \subseteq X\}$.*
2. *Define the operation Θ as $S \Theta J = \{X \setminus J : X \in S\}$.*
3. *Define the classes $\Sigma_0, \Sigma_1, \Sigma_2, \ldots$ recursively as follows.*
   (a) *$S \in \Sigma_0$ if and only if $X \in S \Rightarrow |X| \leq 1$.*
   (b) *$S \in \Sigma_1$ if and only if $\exists a \in I$ such that*
      i. *$S_{\{a\}} \in \Sigma_0$.*
      ii. *$S \Theta \{a\} \in \Sigma_1$.*
   (c) *$S \in \Sigma_k$ if and only if $\exists a \in I$ such that*
      i. *$S_{\{a\}} \in \Sigma_{k-1}$.*
      ii. *$S \Theta \{a\} \in \Sigma_k$.*

Notes 26. *[1]*

1. *if we assume $S = \emptyset \in \Sigma_k \forall k$ then $\Sigma_{k-1} \subseteq \Sigma_k, k = 1, 2, \ldots$.*
2. *$\forall S \exists h$ such that $S \in \Sigma_k \forall k \geq h$.*

Definition 159. ***(k-candidate for a family of sets)***[1]
Let I_S be the set of elements of I that appear in some set of the family S. $a \in I_S$ is a k-candidate for S if and only if $S_{\{a\}} \in \Sigma_{k-1}$ for $k \geq 1$.

Property 49. [1]
For $k \geq 1$, $S \in \Sigma_k$ if and only if $\exists(a_1, a_2, \ldots, a_p)$, where $a_1, a_2, \ldots, a_p$ are all the elements of I_S such that

1. *$S^i_{\{a_i\}} \in \Sigma_{k-1}$, i.e. a_i is a k-candidate for $S^i, 1 \leq i \leq p$.*
2. *$S^{p+1} = \emptyset$, where $S^i = S^{i-1}\Theta\{a_i\}$, and $S^1 = S$.*

Property 50. [1]
For $k \geq 0$ and $a \in I$, $S \in \Sigma_k \Rightarrow S\Theta\{a\} \in \Sigma_k$.

Corollary 4. [1]
For $k \geq 0$ and any $a_1, a_2, \ldots, a_q \in I, q \geq 1$, $S \in \Sigma_k \Rightarrow S\Theta\{a_1, a_2 \ldots, a_q\} \in \Sigma_k$.

The recognition algorithm

In this section we introduced the recognition algorithm for the class Σ_k [1].
In order to describe the algorithm, we need the following definitions.

Definition 160. [1]

1. ***(The labeled rooted tree T(S,k)).***
   Let S be a family of sets and k be an integer.
   Each node of T(S,k) corresponds to a subset of S and the root corresponds to S.
   If a node x corresponds to S_J and its depth is less than k then for each $e \in I$ where $e \in S_J$ there is an edge (x,y) labeled e, where y corresponds to $S_{J\cup\{e\}}$.

   The leaves of the tree are nodes where:

   (a) *The corresponding S_J equal $\emptyset$ or*
   (b) *their depth equal k.*

2. ***(The operation TEST(x)).***
   Let x be a leaf, TEST(x) checks whether the corresponding family is in Σ_0.

3. ***(The operation PRUNE(x)).***
   Let x be a node then PRUNE(x) deletes the element e which labels the edge (p(x),x) [where p(x) is the father of x in the tree] from all the sets of the family S_J corresponding to p(x).

4. *Let L be the list of leaves of T(S,k).*

   (a) ***(The operation HEAD(x,L)).***
   HEAD(x,L) finds the head x of L and deletes it from the list.
   (b) ***(The operation INSERT(x,L)).***
   INSERT(x,L) inserts each leave of the subtree rooted at p(x) in L if it is not already presented in L.

Theorem 18. [1].
Let Σ be a CNF with n variables, the time complexity of the algorithm 10 is within $O(|\Sigma| \times n^k)$.

Algorithm 10: FASTMEMBER(S, k) [1]

```
BUILD(T(S,k),L);
repeat
    HEAD(x,L);
    if x is the root T(S,k) then
        return YES ;
    if TEST(x) then
        begin
            PRUNE(x);
            INSERT(x,L);
until L=nil;
return;
```

Polynomial classes of satisfiability problems

Now we have the necessary material to present the hierarchy Γ_k.

Notation 11. *[1].*
Let P be a set of n Boolean propositions and let T denote the proposition that is always TRUE and F denote the proposition that is always FALSE.
Write a clause in a CNF as follows:
$a_{r+1} \wedge a_{r+2} \wedge \cdots \wedge a_q \rightarrow a_1 \vee a_2 \vee \cdots \vee a_r$ *where* $a_i \in P$, $i = 1, 2, \ldots, q$.
The disjunction $a_1 \vee a_2 \vee \cdots \vee a_r$ is called consequence and the conjunction $a_{r+1} \wedge a_{r+2} \wedge \cdots \wedge a_q$ is called implicant.
Let $X_C = \{a_1, a_2, \ldots, a_r\}$ denote the set of propositions which appear in the consequence.
Let $G = G_H \cup G_N$ be a set of clauses where;

1. *G_H is a subset of the Horn clauses.*
2. *G_N is a subset of the non-Horn clauses.*

Definition 161. *[1].*
Let $S(G) = \{X_C : C \in G\}$.
For $k = 1, 2, \ldots$ let Γ_k be the set of all the instances of SAT for which $S(G) \in \Sigma_K$.

Remark 18. *[1].*

1. $\Gamma_0 = HORN$.
2. $\Gamma_{k-1} \subseteq \Gamma_k, k = 1, 2, \ldots$
3. $SAT = \bigcup_{k=0}^{\infty} \Gamma_k$.

Theorem 19. *[1].*
For any k,Γ_k can be solved in $O((|\Sigma| \times n^k)$ time.

k-resolution

Buning[112] introduced the so-called k-resolution and showed that k-resolution is complete and sound for Γ_{k-1} and it is incomplete for Γ_k.

Definition 162. *(k-resolution) [112].*
The k-resolution is the ordinary resolution but with at least one of the parent clauses is a k-clause.

Theorem 20. *[112].*
A k-resolution is complete and sound for Γ_{k-1}.

3.6.2 The hierarchy of Dalal and Etherington [2].

The hierarchy of Gallo and Scutella has the following restriction:

- There exists $\Sigma \in$2-CNF such that $\Sigma \notin \Gamma_k$ for any integer k. But this is not necessary because 2-CNF can be solved in linear time.

Dalal and Etherington [2] avoid this restriction by giving a hierarchy $\Omega = \Omega_0, \Omega_1, \ldots$ where

1. $(HORN \cup 2 - CNF) \subset \Omega_0$.
2. For each k, j we have $\Gamma_k \subset \Omega_k$ but $\Omega_k \not\subseteq \Gamma_j$.
3. For each Ω_k, the satisfiability problem is solvable in $O((|\Sigma| \times n^k)$ time.

They also show that the two hierarchies differ by a CNF which is not 2-CNF.

Definition 163. *(Multiset and multisubset) [2].*

1. *A multiset is a set but the multiple occurrence of an element in it is permitted.*
2. *A multiset R is multisubset of multiset S denoted by $R \sqsubseteq S$ if and only if for each element x, the number of occurrences of x in R is equal to or fewer than the number of occurrences of x in S.*

Notes 27. *[2]*

1. *Let S be a multiset and let $set(S)$ be the set obtained from S by removing multiple occurrences of elements.*
   Let S and R be multisets, $R \sqsubseteq S \Rightarrow set(R) \subseteq set(S)$.
2. *If S is a multiset then $\{x \in S : P(x)\}$ does not remove duplicate elements.*

Here, HORN and BINARY denote the set of Horn CNFs and the set of binary CNFs, respectively. Let $\Sigma_x = \Sigma|_x$ and Σ^* be as defined in section 4.1.

Definition 164. ***The hierarchies Δ and Ω*** *[2].*
Let $I(\Sigma)$ be the set of literals that occur in the clauses of Σ. The classes Δ_k and Ω_k are defined recursively as follows.

1. *$\Sigma \in \Delta_0$ if and only if either $\perp \in \Sigma$, or Σ contains no positive clause, or Σ contains no negative clause, or there is a unit clause $\{x\} \in \Sigma$ such that $\Sigma_x \in \Delta_0$.*
2. *For any k, $\Sigma \in \Omega_k$ if and only if either $\Sigma \in \Delta_k$ or for all literals $x \in I(\Sigma)$, either $\Sigma^*_x \in \Delta_k$ and $\Sigma^*_{\neg x} \in \Omega_k$ or $\Sigma^*_x \sqsubseteq \Sigma$ and $\Sigma^*_x \in \Omega_k$.*
3. *For any $k > 0$, $\Sigma \in \Delta_k$ if and only if either $I(\Sigma) = \emptyset$ or there is a literal $x \in I(\Sigma)$ such that either $\Sigma^*_x \in \Omega_{k-1}$ and $\Sigma^*_{\neg x} \in \Delta_k$ or $\Sigma^*_x \sqsubseteq \Sigma$ and $\Sigma^*_x \in \Delta_k$.*

Δ denotes the hierarchy $\Delta_0, \Delta_1, \ldots$ and Ω denotes the hierarchy $\Omega_0, \Omega_1, \ldots$

Notes 28. *[2]*

1. *Δ_0 is the set of formulas whose satisfiability can be determined without any case analysis.*
2. *The class Ω_k contains the formulas in which propagating a truth value for any of their literals reduces them to formulas in either Ω_k or Δ_k.*
3. *The class Δ_k, contains the formulas in which propagating the truth of some literal reduces them to formulas in either Ω_{k-1} or Δ_k.*

Theorem 21. *[2].*

1. *For any k,* $\Delta_k \subset \Omega_k \subset \Delta_{k+1}$.
2. *For any class C in* Δ *or* Ω*, if there are formulas* Σ *and* Σ' *such that* $\Sigma \in C$ *and* $set(\Sigma') \subseteq set(\Sigma)$ *then* $\Sigma' \in C$.
3. *HORN*$\subset \Delta_0$ *and 2-CNF*$\subset \Omega_0$.

Dalal and Etherington [2] presented an algorithm that solves the recognition and satisfiabilty problems in Ω_k in $O((|\Sigma| \times n^k)$ time.

Comparison with the hierarchy of Gallo and Scutella

Dalal and Etherington [2] gave the following different notation to the hierarchy of Gallo and Scutella.

Definition 165. *[2]*
For any clause, C, let $C^+ = \{l : l$ *a positive literal,* $l \in C\}$. *For a CNF* Σ*, let* $\Sigma^+ = \{C^+ : C \in \Sigma\}$.
Let $I^+(\Sigma)$ *be the set of all positive literals occurring in the clauses of the CNF* Σ. *For any CNF* Σ*, and any subset J of* $I^+(\Sigma)$*, let*

1. $\Sigma_J = \Sigma \setminus \{C \in \Sigma : J \cap C \neq \emptyset\}$.
2. $\Sigma \Theta J = \{C \setminus J : C \in \Sigma\}$.

$\Gamma = \Gamma_1, \Gamma_2, \ldots$ *are defined as follows:*

1. $\Sigma \in \Gamma_0$ *if and only if* $C \in \Sigma^+ \Rightarrow |C| \leq 1$.
2. *For* $k > 0$, $\Sigma \in \Gamma_k$ *if and only if* $\exists a \in I^+(\Sigma)$ *such that* $\Sigma_{\{a\}} \in \Gamma_{k-1}$ *and* $\Sigma \Theta \{a\} \in \Gamma_k$.

Then they proved the following theorem that explains the relationship between the hierarchy Γ and the hierarchy Ω.

Theorem 22. *[2]*
For each $k, j, \Gamma_k \subset \Omega_k$ *but* $\Omega_k \not\subseteq \Gamma_j$.

3.6.3 The hierarchy of Pretolani [3]

Now we present Pretolani's hierarchy [3] which is a general approach that generalizes the Gallo and ScutelIa's one.
Here we refer to the CNF Σ as $\Sigma = (P, M)$, where P is the set of the propositional variables in Σ and M is the set of the clauses in Σ.

Definition 166. *[3]*
Let $\Sigma = (P, M)$ *be a CNF then*

1. $NS(P) = \{C \in M : \neg p \in C\}$.
2. $PS(P) = \{C \in M : p \in C\}$.
3. $M \setminus p = \{C \setminus \{p, \neg p\} : C \in M\}$.
4. $M_p = \{C \setminus \{\neg p\} : C \in M \setminus PS(p)\}$.
5. $M_{\neg p} = \{C \setminus \{p\} : C \in M \setminus NS(p)\}$.
6. $\Sigma \setminus p = (P \setminus \{p\}, M \setminus p)$.
7. $\Sigma_p = (P \setminus \{p\}, M_p)$.

8. $\Sigma_{\neg p} = (P \setminus \{p\}, M_{\neg p})$.

Definition 167. *[3]*
Let Φ be a class of CNFs.

1. ***(A closed under fixing class).*** *Φ is closed under fixing if and only if when $\Sigma = (P, M) \in \Phi$ then both $\Sigma_p \in \Phi$ and $\Sigma_{\neg p} \in \Phi$ for each $p \in P$.*

2. ***(A closed under union class).*** *Φ is closed under union if and only if when $\Sigma_1 = (P, M_1) \in \Phi$ and $\Sigma_2 = (P, M_2) \in \Phi$ then $\Sigma = (P, M_1 \cup M_2) \in \Phi$.*

3. ***(A closed under splitting class).*** *Φ is closed under splitting if and only if when $\Sigma = (P, M) \in \Phi$ then $\Sigma' = (P, M') \in \Phi$ for each $M' \subseteq M$.*

Example 24. *[3]*
Horn, RHorn, Tovey and binary formulas are closed under fixing and splitting, but only Horn and binary formulas are closed under union.

Pretolani's study of Gallo and Scutella's hierarchy

Pretolani studied Gallo and Scutella's hierarchy and he introduced a recognition algorithm for membership of Γ_k that improved their recognition algorithm.

Definition 168. *[3]*
A formula $\Sigma = (P, M)$ is GHorn=$S_0 = \Gamma_1$ if and only if either Σ is Horn, or there exists a variable $p \in P$ such that

1. *Σ_p is a Horn CNF.*

2. *$\Sigma \setminus p$ is a GHorn CNF.*

The variable p in the this definition is a candidate for Σ; a CNF may have more than one candidate.

Definition 169. *A formula $\Sigma = (P, M)$ belongs to Γ_i if and only if either $\Sigma \in \Gamma_{i-1}$ or there exists a variable $p \in P$ such that*

1. *Σ_p belongs to Γ_{i-1}.*

2. *$\Sigma \setminus p$ belongs to Γ_i.*

Using the techniques from [74] and [1], Pretolani [3] improved the time efficiency of the Γ_i-recognition algorithm.

Theorem 23. *[3]*
For each $i > 0$, Γ_i-recognition can be solved in $O(|\Sigma|\, n^{i-1})$ time.

The general hierarchy of Pretolani [3].

Pretolani [3] has presented a general hierarchy with any polynomially solvable base class (under the quite weak requirement that this base class be closed under fixing, see definition 3.6.3).

Definition 170. ***The general hierarchy*** *Π_i [3].*
Let Π be any polynomially solvable subclass of SAT.
Let $\Pi = \Pi_0$ and for each $i > 0$, a CNF $\Sigma = (P, M) \in \Pi_i$ if and only if either $\Sigma \in \Pi_{i-1}$ or there exists a variable $p \in P$ such that one of the following conditions holds.

1. *$\Sigma_p \in \Pi_{i-1}$ and $\Sigma_{\neg p} \in \Pi_i$;*

2. *$\Sigma_{\neg p} \in \Pi_{i-1}$ and $\Sigma_p \in \Pi_i$.*

Notes 29. *1.* $\Pi_{i-1} \subseteq \Pi_i$.

2. For any CNF Σ *that has n variables we have* $\Sigma \in \Pi_{n-1}$.

Pretolani [3] introduced an algorithm that solves the satisfiability problem for the classes Π_i in $n_i T$ time where T is the time of an algorithm that solves the satisfiability problem for the classes Π_0.
Also, Pretolani [3] introduced an algorithm that recognizes the membership of the classes Π_i in $O(n_i(|\Sigma| + R(\Sigma)))$ time where $R(\Sigma)$ is the time complexity of an algorithm that recognizes the membership of the classes Π_0.
These two algorithms hold under the quite weak requirement that this base class is closed under fixing; this follows from the following theorem.

Theorem 24. *[3]*
If the base class Π_0 *is closed under fixing then each class* Π_i, $i > 0$, *is closed under fixing.*

Split-Horn class and Split-Π class

Pretolani [3] presented a new tractable class that he called Split-Horn. This class is closed under fixing. Let us denote this class by SpHorn. We have the following theorem [3].

Theorem 25. *[3]*
If $\Sigma = (P, M) \in$*SpHorn then both* $\Sigma_p \in$*SpHorn and* $\Sigma_{\neg p} \in$*SpHorn for each* $p \in P$.

Let Σ^p and $\Sigma^{\neg p}$ be the CNFs consisted of the non-Horn clauses of Σ_p and $\Sigma_{\neg p}$, respectively.
The definition of Split-Horn class is the following one [3].

Definition 171. *[3]*
$\Sigma = (P, M) \in$*SpHorn if and only if either* $\Sigma \in$*HORN or there exists a variable* $p \in P$ *such that*

1. $\forall C \in M$, $C \cap \{p, \neg p\} = \emptyset \Rightarrow C$ *is a Horn clause.*

2. Σ^p *and* $\Sigma^{\neg p}$ *belong to SpHorn.*

The satisfiability problem for SpHorn and SpHorn-recognition can be solved in polynomial time[3].

Theorem 26. *let* $\Sigma = (P, M) \in$*SpHorn be a CNF.*

1. The satisfiability problem for Σ *can be solved in* $O(|\Sigma|^2)$ *time.*

2. SpHorn-recognition can be solved in $O(n|\Sigma|)$ *time and* $O(|\Sigma|)$ *space.*

Since SpHorn is closed under fixing, it can be a base class Π_0 for the Π_i hierarchy. Actually (see [3]) we can use any Split-Π where the class Π is closed under fixing, union and splitting, in this case Split-Π is closed under fixing, so Split-Π can be a base class Π_0 for the Π_i hierarchy.

3.6.4 The two hierarchies of Cepek and Kucera [4].

Cepek and Kucera [4] introduced two hierarchies using an approach similar to the general approach of Pretolani [3] (see section 3.6.3). To define these two hierarchies, let us denote the class of renamable-Horn (see section 3.3.2) by *HH* and the set of literals of a CNF by L.

Definition 172. *[4]*
Let Σ *be a formula and* $J \subseteq L$ *be a set of literals that contains no complementary pair.*

1. $\Sigma[J := 0]$ *is the formula obtained from* Σ *by substituting the value zero for all literals from the set J (and the value one for their complements),*

2. $\Sigma[J := 1]$ *is the formula obtained from* Σ *by substituting the value one for all literals from the set J (and the value zero for their complements).*

If $J = \{e\}$, we simply write $\Sigma[e := 0]$ instead of $\Sigma[\{e\} := 0]$ and $\Sigma[e := 1]$ instead of $\Sigma[\{e\} := 1]$.

Definition 173. ***The first hierarchy of Cepek and Kucera [4].***
The hierarchy $\Pi_k, k = 0, 1, \ldots$ is recursively defined by

1. $\Pi_0 = HH$.
2. $\forall k > 0$, $\Sigma \in \Pi_k$ *if and only if* $L = \emptyset$ *or there exists* $e \in L$ *such that*
   (a) $\Sigma[e := 1] \in \Pi_{k-1}$
   (b) $\Sigma[e := 0] \in \Pi_k$

Notes 30. *[4].*

1. *Let $k \geq 0$ be an integer. We have that $\Pi_k \subsetneq \Pi_{k+1}$.*
2. *Let Σ be a formula on n variables. We have that $\Sigma \in \Pi_n$.*
3. $\bigcup_{k=0}^{\infty} \Pi_k$=*SAT.*
4. *Let $k \geq 0$ be an integer, Σ be a CNF and e be a literal. $\Sigma \in \Pi_k$ implies $\Sigma[e := 0] \in \Pi_k$ and $\Sigma[e := 1] \in \Pi_k$.*
5. *Let $k, l \geq 0$ be integers and let $\Sigma \in \Pi_k$ and $\Sigma' \in \Pi_l$ be CNFs on n_Σ and $n_{\Sigma'}$ variables, respectively, with $var(\Sigma) \cap var(\Sigma') = \emptyset$. $\Sigma'' = \Sigma \wedge \Sigma' \in \Pi_{k+l}$, where Σ' is a CNF on $n_\Sigma + n_{\Sigma'}$ variables. Moreover, if $k = min\{i : i \geq 0 \wedge \Sigma \in \Pi_i\}$ and $l = min\{i : i \geq 0 \wedge \Sigma' \in \Pi_i\}$ then $k + l = min\{i : i \geq 0 \wedge \Sigma'' \in \Pi_i\}$.*

Theorem 27. *The recognition and the satisfiability problem for $\Sigma \in \Pi_k$ can be solved in time $O(|\Sigma| \times n^{k+1})$ for any fixed $k \geq 0$.*

Relationship of the hierarchy of Gallo and Scutella and Π hierarchy [4]

In this section we present the relationship between the hierarchy of Gallo and Scutella and the Π hierarchy [4].

Property 51. $\forall k \geq 0, \Gamma_k \subsetneq \Pi_k$.

Definition 174. *[4].*
Let $\overline{\Gamma_k} = \{\Sigma : \exists S \subseteq \{1, 2, \ldots, n\}, \Sigma^S \in \Gamma_k\}$, (see definition 99 for Σ^S).

Property 52. *[4].*
$\forall k \geq 0, \overline{\Gamma_k} \subseteq \Pi_k$.

Property 53. *[4].*
$\overline{\Gamma_0} = \Pi_0$ *and* $\overline{\Gamma_k} \subsetneq \Pi_k, \forall k \geq 1$.

To give the next theorem, we need the following definition.

Definition 175. ***Being closed under disjoint union class.***
A class Φ is closed under disjoint union if and only if when $\Sigma_1 \in \Phi$ and $\Sigma_2 \in \Phi$ are two CNFs with disjoint sets of variables then $\Sigma_1 \wedge \Sigma_2 \in \Phi$.

Property 54. *[4].*
Let Φ be a class of CNFs closed under disjoint union.
If $\Phi \setminus \Pi_0 \neq \emptyset$ then $\Phi \setminus \Pi_k \neq \emptyset, \forall k \geq 0$.

Corollary 5. *[4].*
For each $k \geq 0$ there is 2-SAT CNF $\Sigma_1 \notin \Pi_k$, q-Horn CNF $\Sigma_2 \notin \Pi_k$ and an extended hidden Horn CNF $\Sigma_3 \notin \Pi_k$.

Proof. [4].
Indeed, all three classes are closed under disjoint union and in each one of them there are CNFs which are not hidden Horn. □

The choice of hidden Horn class as a base class in the definition 173 is not essential, what essential for the base class is

1. closeness under variable complementation; and
2. closeness under partial assignment.

So, in the second hierarchy they used q-Horn as base class.

Definition 176. ***The second hierarchy of Cepek and Kucera [4].***

The hierarchy Υ_k, $k = 0, 1, \ldots$ is recursively defined by

1. $\Upsilon_0 = q - Horn.$
2. $\forall k > 0$, $\Sigma \in \Upsilon_k$ *if and only if* $L = \emptyset$ *or there exists* $e \in L$ *such that*
   (a) $\Sigma[e := 1] \in \Upsilon_{k-1}$
   (b) $\Sigma[e := 0] \in \Upsilon_k$

One can prove that almost all results for Π-hierarchy is also TRUE for Υ-hierarchy.
Also we have the following.

Property 55. *[4].*
$\Pi_k \subsetneq \Upsilon_k, \forall k \geq 0.$

Corollary 6. *[4].*
The class Υ_1 properly contains both the class S_0 and the class of q-Horn formulas (i.e., class Υ_0). Moreover, recognition of Υ_1 and satisfiability for Υ_1 can be done in $O(l \times n^2)$ time where l is the length of a CNF in Υ_1.

3.6.5 The Kullmann's hierarchy [5]

Kullmann [5] gave a more general approach than Pretolani's one [3] and improved it in several aspects. To present the Kullmann's hierarchy, we need some notations.

Notation 12. *[5]*

1. *CLS denotes the set of all CNFs*
2. *$\{SAT\}$ denotes the set of satisfiable CNFs and $\{USAT\}$ denotes the set of unsatisfiable CNFs.*
3. *PASS denotes the set of all partial assignments.*
4. *If Σ is a CNF and Φ is a partial assignment then $\Sigma|_\Phi$ denotes the act of Φ on Σ, i.e., $\Sigma|_\Phi$ obtained from Σ by eliminating all clauses containing a literal x assigned truth value 1 by Φ and cancelling all literals assigned truth value 0 by Φ from the remaining clauses of Σ.*
   In particular if v_i are variables and $\epsilon_i \in \{0, 1\}$, $i = 1, 2, \ldots, m$ then
   $\Sigma|_{<v_1 \to \epsilon_1, v_2 \to \epsilon_2, \ldots, v_m \to \epsilon_m>}$ denotes the action of eliminating all clauses that contain a literal $v_i^{\epsilon_i}$, $i = 1, 2, \ldots, m$ and canceling all literals $v_i^{\neg\epsilon_i}$, $i = 1, 2, \ldots, m$ from the remaining clauses of Σ. where $v^1 = v$ and $v^0 = \neg v$.

5. *Let Σ be a CNF and Φ be a partial assignment then $var(\Sigma)$ and $var(\Phi)$ denote the set of the variables of Σ and Φ, respectively.*

6. *$mod_p(\Sigma) := \{\Phi \in PASS : var(\Phi) \subseteq var(\Sigma) \wedge \Sigma|_\Phi = TRUE\}$ is the set of all partial assignments satisfying Σ (the set of all partial models).*

7. *$mod_t(\Sigma) := \{\Phi \in PASS : var(\Phi) = var(\Sigma) \wedge \Sigma|_\Phi = TRUE\}$ is the set of all total assignments satisfying Σ (the set of all total models).*

The $G_k(U, S)$ hierarchy [5]

We present in this section the hierarchy $G_k(U, S)$ [5]. We need the following definitions.

Definition 177. ***enforced assignment***[5].
A partial assignment $\Phi \in PASS$ is an enforced assignment for a CNF Σ if and only if for all $v \in var(\Phi)$ flipping the value of Φ on v yields an unsatisfiable CNF, i.e.,
$\Sigma|_{<v \to \neg\Phi(v)>} \in \{USAT\}$.

Example 25. *If $\{x_1\}, \{x_2\}, \ldots, \{x_r\}$ is the unit clauses of a CNF Σ then $< x_1 \to 1, x_2 \to 1, \ldots, x_r \to 1 >$ is an enforced assignment for Σ.*

Definition 178. ***Stable under enforced assignments class***[5].
A class $C \subseteq CLS$ is stable under enforced assignments if and only if for any $\Sigma \in C$ and any enforced assignment Φ for Σ we have $\Sigma|_\Phi \in C$.

Notes 31. [5]
Let $\Sigma \in CLS$ and $\Phi, \Phi' \in PASS$ then

1. *The following are equivalent.*
   (a) *Φ enforced assignment for Σ.*
   (b) *$\forall \Phi' \in mod_p(\Sigma)$, $\Phi \subseteq \Phi'$.*
   (c) *$\forall \Phi' \in mod_t(\Sigma)$, $\Phi \subseteq \Phi'$.*

2. *If Φ enforced assignment for Σ then $\Sigma|_\Phi$ is equivalent to Σ.*

3. *If Φ enforced assignment for Σ and $\Phi' \subseteq \Phi$ then Φ' enforced assignment for Σ.*

4. *For $\Sigma \in \{USAT\}$ every partial assignment Φ is enforced, so $C \subseteq \{USAT\}$ is stable under enforced assignments if and only if C is stable under partial assignments.*

5. *For any family $(C_i), i \in I$ of classes $C_i \subseteq CLS$ such that all C_i are stable under enforced assignments then $\bigcup_{i \in I} C_i$ and $\bigcap_{i \in I} C_i$ are stable under enforced assignments.*

Also we need the following definition to preset the hierarchy $G_k(U, S)$.

Definition 179. ***Allowing substitution class***[5]
A class $C \subseteq CLS$ allows substitution if and only if for every $\Sigma \in C$ with $var(\Sigma) \neq \emptyset$ there is $(v, \epsilon) \in var(\Sigma) \times \{0, 1\}$ with $\Sigma|_{<v \to \epsilon>} \in C$.

Notes 32. [5].

1. *If $C \subseteq \{USAT\}$ is stable under partial assignments then C trivially allows substitution. But a satisfiable CNF needs not have any enforced assignments, and thus $C \subseteq \{SAT\}$ which is stable under enforced assignments may not allow substitution.*

2. *For any family (C_i), $i \in I$ of classes $C_i \subseteq CLS$ such that all C_i allow substitution then $\bigcup_{i \in I} C_i$ allows substitution too.*

Now we are ready to present the hierarchy $G_k(U,S)$.

Definition 180. ***The hierarchy*** $G_k(U,S)$*[5]*
Consider the CLSs $U \subseteq \{USAT\}$ *and* $S \subseteq \{SAT\}$ *that satisfies.*

1. $U_0 := \{\Sigma \in CLS : \bot \in \Sigma\} \subseteq U$.
   $S_0 := \{\{\}\} \subseteq S$.
2. U *and* S *are stable under enforced assignment and allow substitution.*

Now,

1. *Let* $G^0_0(U,S) := U$.
   Define the classes $G^0_{k+1}(U,S), k \geq 0$ *as follows.*
   $\Sigma \in G^0_{k+1}(U,S)$ *if and only if either* $\Sigma = \bot$ *or there is* $(v,\epsilon) \in var(\Sigma) \times \{0,1\}$ *with* $\Sigma|_{<v \to \epsilon>} \in G^0_k(U,S)$ *and* $\Sigma|_{<v \to \neg\epsilon>} \in G^0_{k+1}(U,S)$.
2. *Let* $G^1_0(U,S) := S$.
   Define the classes $G^1_{k+1}(U,S), k \geq 0$ *as follows.*
   $\Sigma \in G^1_{k+1}(U,S)$ *if and only if either* $\Sigma = \{\}$ *or there is* $(v,\epsilon) \in var(\Sigma) \times \{0,1\}$ *with* $\Sigma|_{<v \to \epsilon>} \in G^1_k(U,S)$ *or* $[\Sigma|_{<v \to \epsilon>} \in G^0_k(U,S)$ *and* $\Sigma|_{<v \to \neg\epsilon>} \in G^1_{k+1}(U,S)]$.

Finally $G_k(U,S) := G^0_k(U,S) \cup G^1_k(U,S)$.

Notes 33. *[5]*

1. *Let* n *be the number of variables of a CNF* Σ *then* $\Sigma \in G_n$.
2. $\bigcup_{k \in N} G^0_k(U,S) = \{USAT\}$ *and* $\bigcup_{k \in N} G^1_k(U,S) = \{SAT\}$.
3. *If* S, S', U, U' *are sets such that*
   (a) $U_0 := \{\Sigma \in CLS : \bot \in \Sigma\} \subseteq U \subseteq U'$.
   $S_0 := \{\{\}\} \subseteq S \subseteq S'$.
   (b) U *and* S *are stable under enforced assignment and allow substitution.*

   and $\epsilon \in \{0,1\}$ *then* $G^\epsilon_k(U,S) \subseteq G^\epsilon_k(U',S')$.

The hierarchy $G_k(U,S)$ forms a cumulative hierarchy, that is:

Property 56. *[5].*
$G_k(U,S) \subseteq G_{k+1}(U,S)\ \forall k \geq 0$.

Also Kullmann [5] presented a very important result that shows the universal property of the classes of his hierarchy $G_k(U,S)$ that forms a basic for all proofs of inclusion of another hierarchy.

Theorem 28. *[5]*
Let $(S_k), k \in N$ *be any family of classes* $S_k \subseteq CLS$ *of CNFs.*
Assume

1. $S_m \subseteq G_m(U,S)$ *for some* $m \geq 0$.
2. *For* $k > m$ *and* $\Sigma \in S_k \setminus S_{k-1}$ *with* $var(\Sigma) \neq \emptyset$, *we have* $\Sigma \in G_k(U,S)$ *or there is* $(v,\epsilon) \in var(\Sigma) \times \{0,1\}$ *with* $\Sigma|_{<v \to \epsilon>} \in S_{k-1}$ *and* $[\Sigma|_{<v \to \epsilon>} \notin \{SAT\} \Rightarrow \Sigma|_{<v \to \neg\epsilon>} \in S_k]$.

Then $\forall \in N, k \geq m \Rightarrow S_k \subseteq G_k(U,S)$.

Also we have the following about the stability of $G_k(U,S)$.

Remark 19. *[5].*

1. *All $G_k(U,S)$ are stable under enforced assignments and allow substitution.*
2. *If U and S are stable under renaming then so all $G_k(U,S)$.*

Finally Kullmann [5] gave the following result.

Theorem 29. *[5]*
Let U and S be as in the definition 180, if the decisions of $\Sigma \in U$ and $\Sigma \in S$ can be done in polynomial time then membership and SAT decision for each class $G_k(U,S)$ is also polynomial time.

3.6.6 The hierarchies that generalize the SLUR class

The SLUR class (see section 14 for SLUR class) has been generalized into hierarchies of tractable classe by different methods [84, 106, 107, 108] We present in this section three hierarchies that generalize the SLUR class: namely, the $SLUR(i)$ hierarchy[84], the $SLUR^*(i)$ hierarchy [106] and the $SLUR_i$ hierarchy [107, 108].

- $SLUR(i)$ hierarchy[84, 106]
  The $SLUR(i)$ hierarchy [84] is obtained by modifying the SLUR algorithm; this is done by choosing i variables (instead of choosing one variable as in the SLUR algorithm) at each step, if the sequence of nondeterministic choices never gives up on the given CNF then this CNF is in the $SLUR(i)$ class.
  Let Σ be a CNF, the $SLUR(i,\Sigma)$ algorithm [84, 106] works as following;

  1. Selects i variables,
  2. Runs unit propagation on all possible 2^i assignments,
  3. If all assignments produce the empty clause in the first iteration $SLUR(i,\Sigma)$ returns unsatisfiable,
  4. If all assignments produce the empty clause in any of the subsequent iterations $SLUR(i,\Sigma)$ gives up,
  5. If at least one of the assignments does not produce the empty clause, $SLUR(i,\Sigma)$ nondeterministically chooses one of these assignments and continues.

  Definition 181. *($SLUR(i)$ class)[84, 106]*
  Let Σ be a CNF. $\Sigma \in SLUR(i)$ if and only if $SLUR(i,\Sigma)$ algorithm does not give up on Σ.

  Property 57. *[84, 106]*

  1. *$SLUR \subsetneq SLUR(1)$ (since a CNF where the SLUR algorithm gives up after selecting the first variable does not belong to the SLUR class but it belongs to SLUR(1)).*
  2. *For every $i \geq 1$,$SLUR(i) \subsetneq SLUR(i+1)$.*
  3. *If Σ is a CNF with n variables then $\Sigma \in SLUR(n)$.*
  4. *SAT=$\bigcup_{i=1}^{\infty} SLUR(i)$.*
  5. *For fixed i, $SLUR(i)$ is a tractable class of SAT but the time complexity grows exponentially in i.*

  Property 58. *[84]*

  1. *The membership problem for the class SLUR is coNP- complete.*
  2. *For each i, the membership problem for the class SLUR(i) is coNP- complete.*

- $SLUR^*(i)$ hierarchy [106]
  The $SLUR^*(i)$ hierarchy [106] is a restriction of a DPLL algorithm (see section 2.5 for DPLL algorithm), since SLUR algorithm itself can be seen as restriction of a DPLL algorithm because SLUR algorithm gives up instead of backtracking more than one level, so one [106] can generalize this intuition by allowing backtracking not one, but i levels before giving up, in this case we have a hierarchy (the so called $SLUR^*(i)$ hierarchy [106]). To explain this, let illustrate it by means of an example [106].

  Example 26. *Consider the following CNF:*
  $\Sigma = \{\{x_1, \neg x_2\}, \{\neg x_1, x_2\}, \{x_1, x_2, x_3, x_4\}, \{x_1, x_2, \neg x_3, x_4\}, \{x_1, x_2, x_3, \neg x_4\}, \{x_1, x_2, \neg x_3, \neg x_4\}\}$.
  $\Sigma \notin SLUR(2)$. *Note that the literals x_1 and x_2 are equivalent, and if the $SLUR(2)$ algorithm chooses x_1 and x_2 and then chooses the assignment $x_1 = x_2 = 0$ then the $SLUR(2)$ algorithm gives up because the remaining CNF is an unsatisfiable quadratic CNF.*
  Hence $SLUR(2)$ algorithm does not benefit from choosing two variables because they are equivalent, but if $SLUR(2)$ algorithm performs unit propagation after choosing a value for x_1 then it will not choose x_2 as second variable and it will be able to know that the the remaining CNF is unsatisfiable if $x_1 = 0$.
  This is the idea behind $SLUR^*(i)$ hierarchy which consist of

  1. *At each step $SLUR^*(i)$ algorithm chooses i variables one by one. then;*
  2. *$SLUR^*(i)$ algorithm performs unit propagation between each of these choices instead of after all of them, see algorithm 12.*

Algorithm 11: test algorithm $test(\Sigma, k)$[106]

```
Input: A CNF formula Σ, a number of decisions k which remains to be made by the algorithm;
Output: A partial assignment which have not led to an empty clause after k decisions, or UNSAT
if no such assignment exists;
(Σ, t) ← UP(Σ) ;
if Σ contains an empty clause then
    return UNSAT ;
if k=0 then
    return empty assignment ;
e ← an undetermined literal (positive or negative);
t'_1 ← test(Σ_1, k − 1);
if previous test did not return UNSAT then
    return t ∪ t_1 ∪ t'_1 ;
t'_2 ← test(Σ_2, k − 1);
if previous test did not return UNSAT then
    return t ∪ t_1 ∪ t'_2 ;
return UNSAT;
```

Property 59. *[106]*

1. *$SLUR(i) \subseteq SLUR^*(i)$. In fact $SLUR(1) = SLUR^*(1)$ and $SLUR(i) \subsetneq SLUR^*(i)$ for $i > 1$.*
2. *$SLUR^*(i) \subsetneq SLUR^*(i+1)$ for every $i \geq 1$.*
3. *$SLUR(i) \cap SLUR^*(2) \neq \emptyset$ for every $i > 1$.*
4. *If Σ is a CNF with n variables then $\Sigma \in SLUR^*(n)$.*
5. *$SAT = \bigcup_{i=1}^{\infty} SLUR^*(i)$.*

Algorithm 12: $SLUR^*(i)$ algorithm $SLUR^*(i, \Sigma)$[106]

```
Input: A CNF formula Σ with no empty clause ;
Output: A partial truth assignment satisfying Σ,UNSAT, or give-up;
(Σ, t) ← UP(Σ) ;
if Σ contains an empty clause then
    return UNSAT ;
while Σ is not empty do
    t' ← test(Σ, i) ;
    if previous test returned UNSAT then
        if it is the first run of the while cycle then
            return UNSAT;
            else give up;
t ← t ∪ t';
return t ;
```

6. *For fixed i, $SLUR^*(i)$ is a tractable class of SAT but the time complexity grows exponentially in i.*

- $SLUR_i$ hierarchy and UC_i hierarchy [107, 108]
  The authors in [107, 108] on one side generalized $SLUR$ to $SLUR_i$ hierarchy that contains both $SLUR(i)$ hierarchy and $SLUR^*(i)$ hierarchy. On the other side they generalized the UC class to UC_i hierarchy (see definition 188). Then they unify the two approaches by showing that $SLUR_i = UC_i \ \forall i$.
  They argued that $SLUR_i$ hierarchy is the natural limit of these approaches, their argument being based on their proof that $SLUR_i = UC_i$.
  In this section we present the basic definitions and results about these approaches.

  1. $SLUR_i$ hierarchy [107, 108]
     The authors in [107, 108] presented $SLUR_i$ hierarchy that generalizes of a (mathematical) definition of $SLUR$ class.
     To define $SLUR$ class mathematically [107, 108], we need the definition of $SLUR$ transition relation from a CNF to a CNF.

     Notation 13. *In this section, we denote the UP procedure as UP_1.*

     Definition 182. *($SLUR$ transition relation) [107, 108]*
     Let Σ and Σ' be two CNFs, the relation $\Sigma \xrightarrow{SLUR} \Sigma'$ holds if and only if there is $x \in L(\Sigma)$ such that $\Sigma' = \Sigma|x = UP_1(\Sigma \cup x)$ and $\Sigma' \neq \bot$.
     The transitive reflexive closure of the relation $\Sigma \xrightarrow{SLUR} \Sigma'$ is denoted by the relation $\Sigma \xrightarrow{SLUR}_* \Sigma'$.

     Using this transition relation, the authors in [107, 108] presented a mathematical definition of the $SLUR$ class and found a natural generalization of it (i.e., the $SLUR_i$ hierarchy).

     Definition 183. *(mathematical definition of $SLUR$ class) [107, 108]*
     Let Σ and Σ' be two CNFs and Let $slur(\Sigma) := \{\Sigma' \in CLS : \Sigma \xrightarrow{SLUR}_* \Sigma' \wedge \neg\exists \Sigma'' \in CLS : \Sigma' \xrightarrow{SLUR}_* \Sigma''\}$, then the $SLUR$ class is defined as $SLUR := \{\Sigma \in CLS : UP_1(\Sigma \cup x) \neq \bot \Rightarrow slur(\Sigma) = \top\}$.

     The generalization of UP_1 that was presented by [107, 108] is the following one.

     Definition 184. *(The generalization UP_i of UP_1 of level i)[107, 108]*
     Let $i \in N \cup \{0\}$, the map $UP_i : CLS \rightarrow CLS$ are

$$UP_0(\Sigma) := \begin{cases} \bot & \text{if } \bot \in \Sigma \\ \Sigma & \text{otherwise} \end{cases}$$

$$UP_{i+1}(\Sigma) := \begin{cases} UP_{i+1}(\Sigma \cup \{x\}) & \text{if } \exists x \in L(\Sigma) : UP_i(\Sigma \cup \{\neg x\}) = \bot \\ \Sigma & \text{otherwise} \end{cases}$$

Now we can define the $SLUR_i$ hierarchy.

Definition 185. *($SLUR_i$ hierarchy)[107, 108]*
Let Σ and Σ' be two CNFs and let $i \in N \cup \{0\}$, the relation $\Sigma \xrightarrow{SLUR:i} \Sigma'$ holds if and only if there is $x \in L(\Sigma)$ such that $\Sigma' = UP_i(\Sigma \cup x)$ and $\Sigma' \neq \bot$.
The transitive reflexive closure of the relation $\Sigma \xrightarrow{SLUR:i} \Sigma'$ is denoted by the relation $\Sigma \xrightarrow{SLUR:i}_* \Sigma'$.
Let $slur_i(\Sigma) := \{\Sigma' \in CLS : \Sigma \xrightarrow{SLUR:i}_* \Sigma' \wedge \neg\exists \Sigma'' \in CLS : \Sigma' \xrightarrow{SLUR:i}_* \Sigma''\}$. The $SLUR_i$ hierarchy is defined as $SLUR_i := \{\Sigma \in CLS : UP_i(\Sigma \cup x) \neq \bot \Rightarrow slur_i(\Sigma) = \top\}$

Notes 34. *[107, 108]*

(a) $SLUR_1 = SLUR$.
(b) $SLUR^*(i) \subset SLUR_{i+1}$ and $SLUR_2 \not\subset SLUR^*(i)$.

2. UC_i hierarchy [107, 108]
In [113] the class UC is presented: it contains the CNFs which have the following property. Deciding if $\Sigma \vDash C$ (i.e., C is an implicate of Σ) can be done by UP.
Using the hardness notion ($hd(\Sigma)$), the authors showed that $hd(\Sigma) \leq i$ if and only if all implicates of Σ can be derived by i-times nested input resolution from Σ. From this, the class UC is the class of CNFs with hardness less than or equal 1. So they define the hierarchy UC_i as the class of CNFs with hardness less than or equal i.

Definition 186. *[107, 108]*
Let Σ be a CNF and $C \in \Sigma$, $C = \{l_1, \ldots, l_k\}$ then $\Sigma \vDash_i C$ if and only if $UP_i(\Sigma \cup \{l_1\} \cdots \cup \{l_k\}) = \bot$.

Definition 187. *(the hardness $hd(\Sigma)$) [107, 108]*
Let Σ be a CNF, the hardness $hd(\Sigma)$ is the minimal integer $i \geq 0$ such that for all clauses C with $\Sigma \vDash C$ we have $\Sigma \vDash_i C$.

Definition 188. *(UC_i hierarchy)[107, 108]*
For $i \geq 0$, $UC_i := \{\Sigma \in CLS : hd(\Sigma) \leq i\}$.

Theorem 30. *[107, 108]*

(a) $SLUR_i = UC_i$ for all $i \geq 0$.
(b) Let Π_i and Υ_i be the two hierarchies defined in section 3.6.4. $\Pi_i \subset UC_{i+1}$ and $\Upsilon_i \subset UC_{i+2}$.

- $CANON(i)$ hierarchy [106, 107, 108]
The canonical CNF of a given a Boolean function is the CNF that consists of all prime implicates of the function. The authors [84] showed that the canonical CNF of a given Boolean function belongs to the SLUR class.

Definition 189. *[84]*

1. *A Boolean function f on n propositional variables $x_1, \ldots, x_n$ is a mapping $f : \{0,1\}^n \to \{0,1\}$.*
2. *let f and g be two Boolean functions, we write $f \leq g$ if and only if $\forall (x_1, x_2 \ldots, x_n) \in \{0,1\}^n$, $f(x_1, \ldots, x_n) = 1 \Rightarrow g(x_1, \ldots, x_n) = 1$. Since a clause is a Boolean function, we have $C_1 \leq f, f \leq C_1$, $C_1 \leq C_2$ (where C_1, C_2 are clauses and f is a Boolean function) as special cases.*

3. *A clause C is an implicate of a Boolean function f if $f \leq C$.*
4. *An implicate clause C of a Boolean function f is prime if $f \leq C$ and $\forall C' \subset C$, $f \not\leq C$.*
5. *A CNF Σ representing a function f is prime if each clause of Σ is a prime implicate of function f.*
6. *The unique CNF consisting of all prime implicates of function f is called the canonical CNF of f.*
7. *A CNF Σ representing a function f is irredundant if removing any clause from Σ produces a CNF that does not represent f.*

Property 60. *[84]*
Let C_1 and C_2 be two clash implicates of a Boolean function f. Then their resolvent $\eta[l, C_1, C_2]$ is also an implicate of f.

Theorem 31. *[84]*
Let f be a Boolean function and let Σ be its canonical CNF, then Σ is a SLUR CNF.

Definition 190. *(derivation of a clause from a CNF by a series of resolutions)[106]*
Let Σ be a CNF that represents a Boolean function f, a clause C can be derived from Σ by a series of resolutions if there is a sequence of clauses $C_1, \ldots, C_k = C$ s.t. for every i, $1 \leq i \leq k$, either $C_i \in \Sigma$ or $C_i = \eta[l, C_{j_1}, C_{j_2}]$ where $j_1, j_2 \leq i$.

Notes 35. *[84]*
Let Σ be a CNF that represents a Boolean function f, every prime implicate of f can be derived from Σ.

Definition 191. *(depth of resolution derivation of a clause from a CNF)[106]*
The depth of resolution derivation of a clause C from a CNF Σ is defined as follows.

1. *It is 0 when $C \in \Sigma$.*
2. *It is the maximum of depths of resolution derivations of C_i and C_j increased by 1, if C can be derived from Σ by a series of resolutions $C_1, \ldots, C_k = C$ where $C = \eta[l, C_i, C_j]$ with $i, j \leq k$.*
3. *It is infinity if and only if C cannot be derived from Σ by a series of resolutions.*

Notes 36. *[106]*
The depth of resolution derivation of a clause from a CNF depends on a particular series of resolutions.

Definition 192. *(resolution depth of a clause with respect to CNF)[106]*
Let C be a clause in a CNF Σ, C has resolution depth d with respect to CNF Σ if and only if C can be derived by a series of resolutions of depth d and there is no series of resolutions of depth smaller than d that can derive C.

Definition 193. *($CANON(i)$ hierarchy)[106]*
Let Σ be a CNF and let f be a Boolean function that is represented by Σ. The CNF $\Sigma \in CANON(i)$, $i \geq 0$ if and only if every prime implicate of f has a resolution depth of at most i with respect to Σ.

Property 61. *[106]*

1. *If $\Sigma \in CANON(0)$ then Σ contains all prime implicates of f where f is a Boolean function represented by Σ.*
2. *Let $\Sigma \in CANON(1)$ and $x \in V(\Sigma)$ then both $\Sigma[:= 0]$ and $\Sigma[:= 1]$ belong to $CANON(1)$.*
3. *If $\Sigma \in CANON(1)$ then either Σ contains an empty clause, or an empty clause is generated during $UP(\Sigma)$.*

The following property shows the relationship between $CANON(i)$ and SLUR, UC_i, $SLUR^*(i)$.

Property 62. *[106, 107, 108]*

1. (a) $CANON(0) \subset SLUR$ *[106].*
   (b) $CANON(1) \subset SLUR$ *[106].*
   (c) $CANON(2) \not\subset SLUR$ *[106].*
2. $CANON(i) \subseteq UC_i$ *and* $UC_1 \not\subseteq CANON(i)$*, so* $CANON(i) \subset UC_i$*[107, 108].*
3. $CANON(2) \cap SLUR^*(i) \neq \emptyset$ *for every* $i \geq 1$*[106].*

3.6.7 The hierarchy of Andrei et al.($Rank_k$ hierarchy)

The authors in [109] introduced a hierarchy ($Rank_k$ hierarchy), where in the subclass of CNFs of rank k any set of k + 1 clauses are related, that is there exists at least one literal in one clause that appears negated in another clause of the set of k + 1 clauses. They showed that the classes $Rank_k$ are tractable classes and these classes are nested and hence they represent a hierarchy.

We first define the reducible and irreducible CNFs.

Definition 194. *(reducible and irreducible CNF)[109]*
A CNF Σ *is reducible if and only if it has two clash clauses, it is irreducible if and only if it is not reducible.*

Definition 195. *(*$dif_V(\Sigma)$*) [109]*

1. *Let* Σ *be a CNF,* $M_V(\Sigma) = \{v \in V(\Sigma) :$ *neither* v *nor* $\neg v$ *appears in any clause of* $\Sigma\}$.
2. $dif_V(\Sigma) = \begin{cases} 0 & \text{if } \Sigma \text{ is reducible} \\ 2^{M_V(\Sigma)} & \text{otherwise} \end{cases}$

The definition of $Rank_k$ hierarchy follows from the definition of $dif_V(\Sigma)$.

Definition 196. *(*$Rank_k$ *hierarchy) [109]*

1. *Let* $\Sigma = \{C_1, C_2, \ldots, C_n\}$ *be a CNF,* Σ *has rank k if and only if* $dif_V(\{C_{i_1}, \ldots, C_{i_{k+1}}\}) = 0$ *for any* $i_1, \ldots, i_{k+1}$ *distinct indices from* $\{1, 2, \ldots, n\}$.
2. $Rank_k$ *denotes the class of all CNFs of rank k.*

The classes $Rank_k$, $k = 1, \ldots$ are nested.

Property 63. *[109]*
$Rank_k \subseteq Rank_{k+1}$.

Example 27. *[109]*
$\Sigma_1 = \{\{x_1, x_2\}, \{\neg x_2, x_3\}\}$ *has rank* 1 *while* $\Sigma_2 = \{\{x_1, \neg x_2\}, \{x_2, \neg x_3\}, \{x_1, x_3\}\}$ *is* $Rank_2$ *CNF but it is not a* $Rank_1$ *CNF.*

The classes $Rank_k$, $k = 1, \ldots$ are tractable.

Property 64. *[109]*
The satisfiability problem for instances in the classes $Rank_k$ *can be decided in polynomial time and checking if a CNF belongs to the classes* $Rank_k$ *can be done in polynomial time.*

3.7 The relationships among Tractable Classes and Hierarchies of Tractable Classes

In this section we give some of the relationships among tractable classes and hierarchies of tractable classes.

Property 65. *(matched, q-Horn and SLUR are mutually incomparable)[77]*

1. $\Sigma = \{\{x_1, x_2, \neg x_4\}, \{\neg x_1, \neg x_3, x_4\}, \{\neg x_2, x_3, \neg x_5\}\}$ *is a matched formula and a SLUR formula but it is not a q-Horn formula .*
2. $\Sigma = \{\{x_1, \neg x_2, x_4\}, \{x_1, x_2, x_5\}, \{\neg x_1, \neg x_3, x_6\}, \{\neg x_1, x_3, x_7\}\}$ *is a matched and a q-Horn formula but it is not a SLUR formula.*
3. *Any Horn formula with more clauses than distinct variables is both SLUR and q-Horn but it is not Matched.*

Property 66. *(LinAut properly contains [q-Horn formula with no unit clauses])[94]*

1. *A q-horn formula without unit clauses is a LinAut formula.*
2. $\Sigma = \{\{x_1, x_2, x_3\}, \{\neg x_1, \neg x_2, \neg x_3\}\}$ *is LinAut formula but it is not a q-Horn formula.*

Property 67. *(SLUR and LinAut are incomparable) [77], [94]*

1. $\Sigma = \{\{\neg x_1, x_2, x_3\}, \{x_1, \neg x_2, x_3\}, \{x_1, x_2, \neg x_3\}, \{\neg x_1, \neg x_2, \neg x_3\}\}$ *is SLUR but not a LinAut formula.*
2. $\Sigma = \{\{x_1, \neg x_2, x_4\}, \{x_1, x_2, x_5\}, \{\neg x_1, \neg x_3, x_6\}, \{\neg x_1, x_3, x_7\}\}$ *is a LinAut formula but it is not a SLUR formula.*

Property 68. *(LinAut properly contains matched formula)[82]*

1. *A matched formula is a LinAut formula.*
2. *Any Horn formula with more clauses than distinct variables and without unit clauses is LinAut (since it is q-Horn) but it is not Matched.*

Property 69. *(nested vs (q-Horn and SLUR)) [77]*

1. $\Sigma = \{\{\neg x_1, x_2, \neg x_3\}, \{\neg x_3, \neg x_4\}, \{x_1, x_3, x_5\}, \{x_3, x_4, \neg x_5\}\}$ *is nested but not q-Horn.*
2. $\Sigma = \{\{x_1, \neg x_2, x_3\}, \{\neg x_1, x_4\}, \{\neg x_1, \neg x_4\}, \{x_1, x_2\}\}$ *is nested but not SLUR.*

Property 70. *(hidden Horn vs S_0)[4]*
hidden Horn formulas is not contained in S_0.

Property 71. *(the relationships among hidden extended Horn,q-Horn and S_0)[4]*

1. $hiddenextendedHorn \cap q - Horn \cap S_0 \neq \emptyset$.
2. $\overline{hiddenextendedHorn} \cap q - Horn \cap S_0 \neq \emptyset$.
3. $hiddenextendedHorn \cap \overline{q - Horn} \cap S_0 \neq \emptyset$.
4. $hiddenextendedHorn \cap q - Horn \cap \overline{S_0} \neq \emptyset$.
5. $\overline{hiddenextendedHorn} \cap \overline{q - Horn} \cap S_0 \neq \emptyset$.
6. $\overline{hiddenextendedHorn} \cap q - Horn \cap \overline{S_0} \neq \emptyset$.
7. $hiddenextendedHorn \cap \overline{q - Horn} \cap \overline{S_0} \neq \emptyset$.

8. $\overline{hiddenextendedHorn} \cap \overline{q-Horn} \cap \overline{S_0} \neq \emptyset$.

Property 72. *(hidden Horn vs the hierarchy of Gallo and Scutella)[4]*
$hiddenHorn \cap (\Gamma_{k+1} \setminus \Gamma_k) \neq \emptyset, \forall k$.

Property 73. *(the relationships among (hidden extended Horn,q-Horn) and the hierarchy of Gallo and Scutella)[4]*

1. $(hiddenextendedHorn \cap q-Horn) \cap (\Gamma_{k+1} \setminus \Gamma_k) \neq \emptyset, \forall k$.
2. $(hiddenextendedHorn \setminus q-Horn) \cap (\Gamma_{k+1} \setminus \Gamma_k) \neq \emptyset, \forall k$.
3. $(q-Horn \setminus hiddenextendedHorn) \cap (\Gamma_{k+1} \setminus \Gamma_k) \neq \emptyset, \forall k$.

With these comparisons between tractable classes and hierarchies of tractable classes, one can conclude that no hierarchy is sufficiently general to include all tractable classes.

Part IV

Contributions

Chapter 4

Extensions and Variants of Dalal's *Quad* Class

This chapter focuses on the so-called *Quad* polynomial fragments proposed in [6] as "almost" quadratic fragments of SAT. Firstly, we establish some properties of *Quad* fragments. Secondly, we extend these fragments and exhibit promising variants, too.

More precisely, we start [114] by studying the sensitivity of *Quad* fragments to clause elimination and variable assignment. Then, an extension is obtained by allowing *Quad* fixed total orderings of clauses to be accompanied with specific additional separate orderings among maximal sub-clauses. Interestingly, the resulting fragments extend *Quad* without degrading its worst-case complexity. Finally, we question other issues founding *Quad* that could be relaxed while keeping the polynomial complexity. Especially, we investigate how bounded resolution and redundancy through unit propagation can play a role in this respect.

The chapter is thus organized as follows. In the next section (4.1) *Quad* classes are presented and we show that Dalal's *Quad* fragments depend on orders in section 4.2. In section 4.3, the sensitivity of *Quad* fragments is analyzed with respect to clauses elimination and variable assignment. In section 4.4, various possible ways to extend *Quad* or lead to variants are listed. In this respect, one extension and one variant of *Quad* are proposed as a case study and analyzed in sections 4.5 and 4.6, respectively.

4.1 The Ordering of Clauses and Quad Classes

This section investigates the so-called *Quad* polynomial fragments of SAT proposed in [6].

We use total orderings $\prec$ between all literals of $\mathcal{L}$. Actually, any total ordering on all literals of $\mathcal{L}$ induces a total ordering among all clauses of $\mathcal{L}$ in the following sense.

Notation 14. *Let $\prec$ be a total ordering among all literals of $\mathcal{L}$ and C and C' be two clauses of $\mathcal{L}$: $C \prec C'$ if and only if $C \subset C'$ or there exists a literal l in $C \setminus C'$ such that $l \prec m$ for each literal m in $C' \setminus C$.*

Quad is based on a linear-time fragment of SAT called *Root*. The *Root* fragment consists of four parts (all four are solved and recognized as polynomial subclasses of SAT);

Definition 197. *(Root) [6]*
A formula Σ is in class Root, if either

1. *$\bot \in \Sigma$, or*
2. *Σ contains no positive clause, or*
3. *Σ contains no negative clause, or*
4. *all clauses of Σ are binary.*

Based on *Root*, *Quad* fragments are defined as follows.

Definition 198. *(Quad) [6]*
Let $\prec$ be a total ordering between literals of $\mathcal{L}$.
A formula Σ is in class Quad if and only if

1. Σ^* *belongs to Root, or*

2. *for the first maximum sub-clause* C' *of the first clause* $C \in \Sigma^*$ *for which* $(\Sigma \wedge \overline{C'})^*$ *is in class Root:*

   (a) *either* $(\Sigma \wedge \overline{C'})^*$ *is satisfiable, or*

   (b) *the formula* $(\Sigma \setminus \{C\}) \cup \{C'\}$ *is in class Quad.*

One can define a subclass of SAT by taking all the possible orders.

Notation 15. *($\bigcup$ Quad)*
As emphasized by Dalal himself, Quad depends on the considered ordering of literals, which induces a total ordering of clauses (and sub-clauses) of Σ*. Different orderings can lead to different Quad fragments see section 4.2. The set of all Quad fragments is noted* $\bigcup$ *Quad.*

Quad is almost quadratic time subclass of SAT.

Remark 20. *Interestingly, a formula* Σ *in Quad can be both recognized and solved in* $O(d \times |\Sigma|^2)$ *time, where* d *is the size of the longest clause in* Σ*. Accordingly, when a fixed upper bound is enforced on the clause length in* Σ*, Quad is recognizable and solvable in quadratic time.*

Tractable algorithms

In this section three tractable algorithms to check the satisfiability of the CNF Σ are presented; these algorithms return either {} or $\bot$ or a simplified CNF Σ' where {} denotes that Σ is satisfiable, $\bot$ denotes that Σ is unsatisfiable. The 2-SAT instances can be solved in linear time; see section 3.3.1. Dalal [6] refers to the algorithm that solves 2-SAT instance Σ as *Binsat*(Σ).

Definition 199. *(Binsat function)*
Binsat(Σ) *returns* Σ *if* Σ *has a non-binary clause else it returns* {} *if* Σ *is satisfiable; it returns* $\bot$ *if* Σ *is unsatisfiable.*

4.1.1 The Basic Algorithm RootSat

The algorithm RootSat (see Algorithm 13) recognizes and solves the elements of the RootSat class. It works as follows.

- Firstly, it repeatedly removes the unit clauses using the UP procedure until we get a CNF without unit clause.
- Then the conditions for membership in class *Root* are tested one by one;
  1. If the resulting CNF from the UP procedure has no positive clauses or no negative clauses, the algorithm RootSat returns {} (i.e., satisfiable).
  2. If the resulting CNF from the UP procedure has positive clauses and negative clauses then it uses *Binsat* to check if it is binary CNF or not and if it is binary then it returns {} (i.e., satisfiable) or $\bot$ (i.e., unsatisfiable), see Algorithm 13.

The algorithm RootSat is complete for the *Root* class and halts in time $O(|\Sigma|)$.

4.1.2 The Qsat algorithm

The algorithm Qsat (see algorithm 14) is a recognition and decision algorithm for the *Quad* class.
It works as follows.

1. It takes each non-unit clause σ of Σ one by one according to the order $\prec$.

Algorithm 13: RootSat(Σ) [6]

```
while Σ has a unit clause do
    select a unit clause {x};
    Σ ← {C ∈ Σ|x ∉ C};
    Σ ← {C \ {¬x}|C ∈ Σ};
if ⊥ ∈ Σ then
    Σ ← ⊥;
if Σ has no positive clause then
    Σ ← {};
if Σ has no negative clause then
    Σ ← {};
return Binsat(Σ);
```

2. After choosing the clause σ in step 1, it takes each max-subclause μ of σ according also to the order $\prec$.

3. It computes R where R is RootSat($\Sigma \cup \neg\mu$). If $R \in Root$ and it is satisfiable then Qsat returns satisfiable; if $R \in Root$ and it is unsatisfiable then Qsat replaces Σ by μ in the CNF Σ and repeats according to step 1 and 2.

4. Qsat terminates in one of two cases:

   - Σ is determined to be satisfiable or unsatisfiable.
   - all μ's and σ's are taken.

Algorithm 14: Qsat(Σ) [6]

```
Σ := RootSat(Σ);
repeat
    if Σ = {} or Σ = ⊥ then
        return Σ;
    σ ← next non-unit clause in Σ;
    μ ← next max-subclause in σ;
    R ←RootSat(Σ ∪ ¬μ);
    if R = {} then
        return {};
    if R = ⊥ then
        Σ ←RootSat(Σ \ ({σ} ∪ {μ});
    reset σ and μ;
until no more σ and μ;
return Σ;
```

Algorithm Qsat is complete for the *Quad* class and halts in $O(|\Sigma|^3)$.

4.1.3 The Quadsat algorithm

The Qsat algorithm suffers from repeating the computations for each σ and μ after each change in Σ. So, in this section we present a more efficient algorithm QuadSat [6] that avoids the previous pitfalls of the Qsat algorithm. The idea can be summarized as follows:

1. Compute R of line 7 almost once for each σ and μ.
2. Modify R of line 7 as Σ changes in line 11.

The total time for each R is $O(|\Sigma|)$, in contrast to $O(|\Sigma|^2)$ used in Qsat. But, there will be a new factor depending on the length of the longest clause.

Algorithm 15: Quadsat(Σ) [6]

```
1  Σ ← RootSat(Σ);
2  repeat
3      if Σ = {} or Σ = ⊥ then
4          return Σ;
5      σ ←next non-unit clause in Σ;
6      μ ←next max-subclause in σ;
7      R(σ, μ) ←RootSat(Σ ∪ ¬μ);
8      if R(σ, μ) = ⊥ or {} then
9          AddQ((σ, μ));
10     Σ ← ProcessQ(Σ);
11 until no more σ and μ;
12 return Σ;
```

ProcessQ

The algorithm ProcessQ inserts the tuples in the queue one by one. We have the following.

1. $R = \{\}$, in this case the given CNF is satisfiable.
2. $R = \bot$, in this case Σ is shortened.
3. If a clause is shortened, we get a new R.
4. Tuples in which R becomes $\bot$ or $\{\}$ are added to the queue.

Theorem 32. *[6]*
For any CNF, algorithm Qsat and QuadSat return the same CNF.

Finally, we have that:

Theorem 33. *[6]*
Let Σ be a CNF, the algorithm QuadSat halts in time $O(|\Sigma|^2 k)$ where k is the length of the longest clause in Σ.

4.1.4 Comparison between *Quad* and q-Horn [6]

Here we show that the classes *Quad* and q-Horn are incomparable by giving an example that belongs to *Quad* but not to q-Horn and an example that belongs to q-Horn but not to *Quad*.

Example 28. *[6]*

1. *For any k, let Σ_K be a K-SAT instance with all clauses of length k built on the set of variables $\{x_1, \ldots, x_k\}$. For any ordering, $\Sigma_K \in$Quad but for any $K > 3$,$\Sigma_K \notin$q-Horn.*
2. *Let $\Sigma = \{\{p, q, r\}, \{\neg p, \neg s, \neg t\}, \{a, b, c\}, \{\neg a, \neg d, \neg e\}\}$.*
   Σ is renamable Horn because it becomes Horn if p, q, a, b are renamed, so it belongs to q-Horn, but $\Sigma \notin$ Quad.

These two examples show that the Quad and q-Horn classes are incomparable.

Algorithm 16: ProcessQ(Σ) [6]

```
while not EmptyQ() do
    (λ, π) ← RemoveQ();
    if λ ∉ Σ then
        continue;
    if R(λ, μ) = {} then
        return {};
    if R(λ, μ) ≠ ⊥ then
        continue;
    Σ ←RootSat((Σ \ ({λ}) ∪ {π});
    p ← λ \ π;
    forall the R(σ, μ) (% ordered using ≺) do
        if σ = λ then
            R(σ, μ) ← R(π, μ − {p}) ←RootSat(Σ ∪ ¬(μ − {p}));
        else if R(σ, μ) has p in a clause ψ obtained from λ then
            R(σ, μ) ←RootSat((R(σ, μ) \ ({ψ}) ∪ ({ψ \ {p}}) ;
        if R(σ, μ) = ⊥ or {} then
            AddQ((σ, μ));
return Σ;
```

4.2 *Quad* Fragments and Orders

The main intuitions behind Definition 198 of the *Quad* class are summarized in the following remark.

Remark 21. *Condition 1) is the base case: it checks whether Σ (closed under unit propagation) belongs to Root, the polynomial base fragment.*
Condition 2.b) expresses a recursive step where it is checked whether a simplified Σ belongs to Quad: a positive answer is sufficient to show that Σ belongs to Quad since (i) one clause C of Σ is replaced by one smaller clause C' to form the simplified instance and (ii) C' has been proved to be a (unit propagation) logical consequence of Σ.
When satisfied, condition 2.a) allows Σ to be classified as belonging to Quad as a more constrained instance (namely, $\Sigma \wedge \neg C'$ that thus contains the additional clauses given by $\neg C'$ before closure by unit propagation) has been shown both satisfiable and belonging to Root, and consequently to Quad.

Quad depends on the considered ordering of literals. Different orderings can lead to different *Quad* fragments, as shown by the following example.

Example 29. *Let $\Sigma = \{\{a,b,c,d,e\},\{a,b,c,d,\neg e\},\{e,a,q,u\},\{e,b,r,v\},\{e,c,s,w\},\{e,d,t,x\},\{\neg f,\neg g\},\{\neg i,\neg j\}\}$. If we consider the literals ordering $\prec$ defined by $a \prec b \prec c \prec e \prec d \prec$ (any ordering about the other literals) then $\Sigma \in$ Quad. Indeed, if we try to deduce the maximum sub-clause $\{a,b,c,e\}$ through unit propagation then the literal d will become true and we will get the formula $\{\{q,u\},\{r,v\},\{s,w\},\{\neg f,\neg g\},\{\neg i,\neg j\}\}$, which is binary and satisfiable. On the contrary, if we adopt the ordering $a \prec b \prec c \prec d \prec e \prec \neg e \prec q \prec u \prec r \prec v \prec s \prec w \prec t \prec x \prec \neg f \prec \neg g \prec \neg i \prec \neg j$ then $\Sigma \notin$ Quad. Indeed, the first max-sub-clause $\{a,b,c,d\}$ of the first clause $\{a,b,c,d,e\}$ can be deduced by unit propagation. Consequently, by substituting $\{a,b,c,d,e\}$ by $\{a,b,c,d\}$, we obtain $\Sigma' = \{\{a,b,c,d\},\{a,b,c,d,\neg e\},\{e,a,q,u\},\{e,b,r,v\},\{e,c,s,w\},\{e,d,t,x\},\{\neg f,\neg g\},\{\neg i,\neg j\}\}$. For all max-sub-clauses of the first clause $\{a,b,c,d\}$ the resulting formula does not belong to Root. Now, we process the second clause $\{a,b,c,d,\neg e\}$. Again, its first max-sub-clause $\{a,b,c,d\}$ can be deduced by unit propagation leading to a new formula $\Sigma'' = \{\{a,b,c,d\},\{e,a,q,u\},\{e,b,r,v\},\{e,c,s,w\},\{e,d,t,x\},\{\neg f,\neg g\},\{\neg i,\neg j\}\}$. One can easily verify that*

for any max-sub-clause c' of any clause $c \in \Sigma''$, the formula $(\Sigma'' \wedge \neg c')^*$ does not belong to Root. So Σ''(and hence Σ) does not belong to Quad.

4.3 About the Stability of *Quad* Fragments

In this section, we highlight properties of *Quad* fragments that play a key role in defining extensions and variants of *Quad*. To the best of our knowledge, these properties of *Quad* have not been described so far.

Let us first analyse the stability of *Quad* with respect to variable assignment and clause removal.

Definition 200. *Let X be a fragment of SAT and μ be an inference process that, given a set of clauses Δ, yields a corresponding set of clauses, noted $\mu(\Delta)$. X is stable with respect to μ if and only if for all $\Delta \in X$, we have that $\mu(\Delta) \in X$.*

Theorem 34. *Quad (resp. $\bigcup$ Quad) is neither stable with respect to clause elimination nor variable assignment.*

The following example suffices to prove this result.

Example 30. *Assume $\Sigma = \{\{a,b,c,d,e\},\{a,b,g\},\ \{a,b,\neg f\},\{c,e,f\},\{d,e,h\},\ \{\neg i,\neg j\},\ \{\neg l,\neg k\}\}$ and consider $\prec$ defined by $a \prec b \prec c \prec d \prec e \prec$ (with any ordering for the other literals). Notice that $\Sigma \in$ Quad. Indeed, if we try to deduce the maximum sub-clause $\{a,b,c,d\}$ through unit propagation then the literal e will become true and we will get the formula $\{\{\neg i,\neg j\},\{\neg l,\neg k\}\}$, which is binary and satisfiable. Consequently, Σ belongs to Quad. Let us consider the formula $\Sigma' = \{\{a,b,g\},\{a,b,\neg f\},\{c,e,f\},\{d,e,h\},\{\neg i,\neg j\},\{\neg l,\neg k\}\}$ obtained by removing the first clause $\{a,b,c,d,e\}$ from Σ. The formula Σ' does not belong to $\bigcup$ Quad. This thus shows that neither Quad nor $\bigcup$ Quad are stable with respect to clause elimination. Now, if we assign a literal d to true, we obtain a new formula $\Sigma'' = \{\{a,b,g\},\{a,b,\neg f\},\{c,e,f\},\{\neg i,\neg j\},\{\neg l,\neg k\}\}$ which does not belong to $\bigcup$ Quad, showing that neither Quad nor $\bigcup$ Quad are stable with respect to variable assignment.*

Remark 22. *The violation of the stability property by clause elimination and literal assignment is not surprising, since the Root class combines several tractable classes (binary clauses, formulas with no positive or no negative clauses) which are founded on some syntactical grounds. In the following, we show that even if the clause to be eliminated is redundant (either in the full case or redundant through UP, as UP is the cornerstone concept of Quad), the remaining formula might cease to belong to Quad.*

The following theorem concerns the stability with respect to elimination of redundant clauses.

Theorem 35. *Quad (resp. $\bigcup$ Quad) is not stable with respect to elimination of redundant clauses*

Proof. The proof follows from Example 30. As we can notice, the removed clause $\{a,b,c,d,e\}$ from Σ is UP-redundant in Σ. □

From the above properties of *Quad* and $\bigcup$ *Quad*, it follows that adding within Σ clauses that are actually redundant in Σ can lead to a logically equivalent set of clauses Σ' that belongs to $\bigcup$ *Quad* whereas Σ does not itself belong to $\bigcup$ *Quad*. Among these redundant clauses, of particular interest are those that can be derived by resolution (namely, resolvents).

Now, obviously, adding any redundant clause within one *Quad* fragment does not guarantee that the resulting set of clauses belongs to $\bigcup$ *Quad*.

All these results show that enhancing *Quad* by means of considering logically equivalent sets of clauses or through processes that consider instantiated formulas are more prone to deliver variant polynomial fragments of SAT than extensions of *Quad*.

4.4 Various Possible Variants

Quad is based on several parameters that we can try to relax while keeping the polynomial-time recognition and satisfiability-testing properties.

4.4.1 Enriching the *Root* fragment

The *Root* class can be enriched with other known polynomial fragments. Obviously, when non linear-time recognizable and solvable fragments are added, this entails an increase in the polynomial worst-case complexity of recognition and satisfiability-checking procedures, due to the *Quad* handling schema. Note that *Quad* already captures several well-known polynomial fragments of SAT that are not fragments of *Root*, like the Horn one. Especially, it seems possible to enrich *Root* with variables renaming features, as well as for example Tovey's class [9] and its variants [115]. We do not explore these paths in this thesis.

4.4.2 Using additional polynomial deductive paradigms

One key feature of *Quad* is closure by Unit Propagation which is also a linear time-process. Likewise, adding or switching to other polynomial-time limited deduction procedures would allow the polynomial feature to be kept. As a case study, we shall compare $\bigcup$ *Quad* with the fragments obtained by replacing unit propagation with restricted forms of resolution like bounded-resolution which has long been known useful in preprocessing steps of SAT solvers (see early works by [116, 117]). We also investigate the extent to which filtering clauses that are unit-propagation redundant ones leads to different SAT fragments.

4.4.3 Relaxing the ubiquitous use of a unique total ordering between clauses

Another question is whether or not the choice of founding *Quad* only on total orderings between all literals prevents or not some polynomial instances from belonging to $\bigcup$ *Quad*. Such a total ordering plays a twofold role in the second point of the definition for *Quad*: it rank-orders both clauses C in Σ^* and the maximum sub-clauses of C. Actually, we shall show that decoupling these two orderings allows more instances to be recognized and solved. We start with this question in the next section.

4.5 *Ext-Quad* fragments

The main idea is to decouple

- the total ordering of clauses that is used in the second item of the definition of *Quad* to examine clauses in Σ^*,
- from the ordering that is used to rank-order the maximal sub-clauses of C' in the same item of the definition of *Quad*. Actually, we allow this latter ordering to differ for the various clauses C'.

Accordingly, the definition of the new fragments, called *Ext-Quad* (for *Extended Quad*), will make use of additional total orderings between sub-clauses that are specific to each clause C'. We note such an ordering $\prec_{C'}$. We define *Ext-Quad* as follows.

Definition 201. *(Ext-Quad)*
Let $\prec$ be a total ordering between clauses of Σ. For each clause C, let $\prec_C$ be a total ordering between the maximum sub-clauses of C.
A formula Σ is in class Ext-Quad if and only if:

1. *Σ^* belongs to Root, or*
2. *for the first (with respect to $\prec_C$) maximum sub-clause C' of the first (with respect to $\prec$) clause $C \in \Sigma^*$ for which $(\Sigma \wedge \neg C')^*$ is in class Root:*
   (a) *either $(\Sigma \wedge \neg C')^*$ is satisfiable, or*
   (b) *the formula $(\Sigma \setminus \{C\}) \cup \{C'\}$ is in class Ext-Quad.*

The following example gives an instance that belongs to Ext-*Quad*.

Example 31.
Let $\Sigma = \{\{p, q, r\}, \{p, q, \neg r\}, \{\neg p, \neg q, \neg r\}\}$. *Since* $\Sigma^* = \Sigma$ *and* $\Sigma \notin Root$, *we have that* $\Sigma^* \notin Root$. *Consider the total ordering* $\prec$ *defined by* $\{p, q, r\} \prec \{p, q, \neg r\} \prec \{\neg p, \neg q, \neg r\}$ *on the clauses of* Σ. *Consider the following total orderings on the max-sub-clauses of the clauses of* Σ *and of some of their sub-clauses:*

- $\prec_{\{p,q,r\}}$ *is defined by* $\{p, q\} \prec_{\{p,q,r\}} \{p, r\} \prec_{\{p,q,r\}} \{q, r\}$
- $\prec_{\{p,q,\neg r\}}$ *is defined by* $\{p, q\} \prec_{\{p,q,\neg r\}} \{p, \neg r\} \prec_{\{p,q,\neg r\}} \{q, \neg r\}$
- $\prec_{\{\neg p,\neg q,\neg r\}}$ *is defined by* $\{\neg p, \neg q\} \prec_{\{\neg p,\neg q,\neg r\}} \{\neg p, \neg r\} \prec_{\{\neg p,\neg q,\neg r\}} \{\neg q, \neg r\}$
- $\prec_{\{p,q\}}$ *is defined by* $\{p\} \prec_{\{p,q\}} \{q\}$

For the first clause $C = \{p, q, r\}$ *and its first max-sub-clause* $C' = \{p, q\}$, *we have that* $(\Sigma \wedge \neg C')^* = \{\bot\} \in Root$. *We can thus now consider* $(\Sigma \setminus \{C\}) \cup \{C'\} = \{\{p, q\}, \{p, q, \neg r\}, \{\neg p, \neg q, \neg r\}\}$. *Consider the ordering* $\{p, q\} \prec \{p, q, \neg r\} \prec \{\neg p, \neg q, \neg r\}$. *When we take the first clause* $C_1 = \{p, q\}$ *according to* $\prec$, *and its first max-sub-clause* $C'_1 = \{p\}$, *we have that* $(((\Sigma \setminus \{C\}) \cup \{C'\}) \wedge \neg C'_1)^* = \{\}$. *Hence,* $\Sigma \in Ext\text{-}Quad$.

Notation 16. *Let us note* $\bigcup Ext\text{-}Quad$ *the set-theoretic union of all Ext-Quad fragments obtained by considering all possible orderings of clauses and sub-clauses.*

The following result shows that $\bigcup Ext\text{-}Quad$ strictly extends $\bigcup Quad$: some instances of *Ext-Quad* are never captured by *Quad*, no matter the ordering between literals used for *Quad*.

Theorem 36.

$\bigcup Quad \subset \bigcup Ext\text{-}Quad$

Proof. It is easy to see that $\bigcup Quad \subseteq \bigcup Ext\text{-}Quad$. Indeed, let Σ be a CNF that belongs to *Quad* with respect to a total ordering $\prec$ on clauses. $\prec$ can also be used as a unique ordering between sub-clauses in the definition of *Ext-Quad*. Hence, Σ also belongs to at least one *Ext-Quad* fragment.

Let us show that the converse does not hold by exhibiting one Σ that belongs to $\bigcup Ext\text{-}Quad$ but to no *Quad* fragment. To this end, we consider the following formula $\Sigma = \{\{a, b, c, d\}, \{a, b\}, \{\neg a, b\}, \{c, e, f\}, \{d, g, h\}, \{\neg i, \neg j\}, \{\neg k, \neg l\}\}$. It is easy to show that Σ belongs to *Ext-Quad* for the ordering of clauses $\{a, b, c, d\} \prec \{a, b\} \prec \{\neg a, b\} \prec \{c, e, f\} \prec \{d, g, h\} \prec \{\neg i, \neg j\} \prec \{\neg k, \neg l\}$ and the constraint that the max-sub-clause $\{a, c, d\}$ of $\{a, b, c, d\}$ precedes the other max-sub-clauses of $\{a, b, c, d\}$. Indeed, $(\Sigma \wedge \overline{\{a, c, d\}})^* = \{\{e, f\}, \{g, h\}, \{\neg i, \neg j\}, \{\neg k, \neg l\}\}$, which belongs to *Root* and is satisfiable; thus, Σ belongs to *Ext-Quad* for this ordering and $\Sigma \in \bigcup Ext\text{-}Quad$. But for any ordering of literals, the clauses $\{c, e, f\}, \{d, g, h\}, \{\neg i, \neg j\}, \{\neg k, \neg l\}$ will not change and the clause $\{a, b\}$ always precedes the clause $\{a, b, c, d\}$ (since $\{a, b\} \subset \{a, b, c, d\}$). We have two cases:

1. $\{a, b\} \prec \{\neg a, b\}$
2. $\{\neg a, b\} \prec \{a, b\}$

In the first (resp. second) case, we have just for the max-sub-clause $\{b\}$ of the clause $\{a, b\}$ (resp. $\{\neg a, b\}$) that $(\Sigma \wedge \overline{\{b\}})^*$ belongs to *Root* and is unsatisfiable, so $\{a, b\}$ (resp. $\{\neg a, b\}$) will change to $\{b\}$ and $(\{\{a, b, c, d\}, \{b\}, \{\neg a, b\}, \{c, e, f\}, \{d, g, h\}, \{\neg i, \neg j\}, \{\neg k, \neg l\}\})^* = \{\{c, e, f\}, \{d, g, h\}, \{\neg i, \neg j\}, \{\neg k, \neg l\}\}$ will never belong to *Quad* (resp. $(\{\{a, b, c, d\}, \{a, b\}, \{b\}, \{c, e, f\}, \{d, g, h\}, \{\neg i, \neg j\}, \{\neg k, \neg l\}\})^* = \{\{c, e, f\}, \{d, g, h\}, \{\neg i, \neg j\}, \{\neg k, \neg l\}\}$ will never belong to *Quad*). Thus, $\Sigma \notin \bigcup Quad$ □

Moreover, the use of (additional) total orderings of sub-clauses does not alter the complexity results obtained for *Quad* in [6]. It is easy to show that,

Theorem 37. *The membership and satisfiability of a CNF* Σ *of size n belonging to a given Ext-Quad fragment can be determined in* $O(|\Sigma|^3)$. *Moreover, a formula* Σ *in Ext-Quad can be both recognized and solved in* $O(d \times |\Sigma|^2)$ *time, where d is the size of the longest clause in* Σ.

Proof. 1. Since the number of different R's that are created by algorithm 15 are at most $|\Sigma|$, we have that in line 13 of algorithm 16, $m-1$ new R's are created for each replacement of a clause λ of length m by a clause π in line 9 of algorithm 16. And since each clause can be shortened to a unit clause in the worst case, we have that at most $\frac{|\Sigma|(d-1)}{2}$ new R's are created in this way.

2. From 1, the total number of created R's is $\frac{|\Sigma|(d+1)}{2}$.

3. From 2 and since the the time of creating each of R's is at most $O(|\Sigma|)$, we have that the total time for creating R's is $O(|\Sigma|^2 d)$.

4. Also at most $\frac{|\Sigma|(d+1)}{2}$ tuples are added to the queue with time at most $O(|\Sigma| dlog|\Sigma|)$ to maintain the queue. And since in each step where Σ is changed in line 9 of algorithm 16 we have that all other R's are accessed, so the total time in this case is $O(|\Sigma|^2)$.

By accumulating all the times, we get that the total time is $O(d \times |\Sigma|^2)$ where d is the size of the longest clause in Σ. So in the worst case the total time is $O(|\Sigma|^3)$. □

In the next section, we turn to another variant of *Quad* obtained by replacing unit propagation by bounded resolution.

4.6 *BR(k)-Quad*

In this section, we investigate the fragments of SAT that are obtained when saturation by unit propagation in the definition of *Quad* is replaced by forms of saturation by bounded resolution. Bounded resolution was introduced in [118]. Saturation by k-bounded resolution consists in adding all resolvents of size less or equal to k. For example, if Σ contains both clauses $a \vee b \vee \neg c \vee d \vee e$ and $a \vee b \vee c \vee e$ then $a \vee b \vee d \vee e$ is a resolvent of size 4. Saturating Σ by 4-bounded resolution will conduct $a \vee b \vee d \vee e$ to belong to the resulting set of clauses. The bound k, called width, is considered as an important complexity measure for resolution refutations [119, 120]. In [119], the authors give a general relationship between the width and the length of a refutation, reducing the problem of giving lower bounds on the length to that of giving lower bounds on the width. More generally, restricting resolution to the derivation of size-bounded resolvents guarantees polynomial complexity. Obviously, such restriction leads to an incomplete resolution proof system.

Several forms of bounded resolution can be considered. These different forms depend on the way according to which the size of the generated resolvents is bounded. One can for example set the size k (bound) of the resolvent between C_i and C_j to $max(|C_i|, |C_j|)$ or to the size of the longest clause of Σ. In these two cases, unit resolution can be interpreted as a special case of bounded resolution.

In our bounded resolution variant, the bound k is set to the size of the longest clause of Σ. Moreover, in our definition of saturation by bounded resolution, subsumed clauses do not take part in the resulting set of clauses. The definition is as follows.

Definition 202. *Let k be the size of the longest clause of* Σ. Σ^{br} *represents the formula* Σ *after closing it by bounded resolution. It is defined recursively as follows:*

1. $\Sigma^{br} = \Sigma$ *if* $\forall C_i = (x \vee \alpha) \in \Sigma$, $\forall C_j = (\neg x \vee \beta) \in \Sigma$, $R = \eta[x, C_i, C_j] \in \Sigma$ *or* $\exists\, C \in \Sigma$ *s.t. C subsumes R.*

2. $\perp \in \Sigma^{br}$ *if* Σ *contains two opposite unit clauses* $\{x\}$ *and* $\{\neg x\}$,

3. *otherwise,* $\Sigma^{br} = (\{R\} \cup \{C \in \Sigma | R \not\subseteq C\})$ *where R is not tautological and is an unsubsumed resolvent between two clauses of* Σ *and* $|R| \leq k$.

The following definition gives the consequence modulo bounded resolution of a clause from a CNF.

Definition 203. *Let* Σ *be a CNF formula. A clause C is a consequence modulo bounded resolution of* Σ, *noted* $\Sigma \models_{BR} C$, *if and only if* $\perp \in (\Sigma \wedge \overline{C})^{br}$

Proposition 1. *If* $\Sigma \models_{UP} C$ *then* $\Sigma \models_{BR} C$.

Proof. It follows from our definition of bounded resolution and of our setting of the bound k to the size of longest clause of the formula. Indeed, UP closure is a special case of BR closure. □

Remark 23. *Not surprisingly, in [118], it is shown that k-bounded resolution is not complete for refutation because k-bounded refutation can be achieved in a polynomial time. If a formula can be refuted by k-bounded resolution then refutation needs at most* $O(n^k)$ *steps of resolution, where n is the number of variables of the CNF formula [121]. This result is direct because no more than* n^k *different resolvents can be generated.*

Let us now define our variant of *Quad* using bounded resolution, noted *BR*(k)-*Quad*.

Definition 204. *(BR(k)-Quad)*
Let $\prec$ *be a total ordering between literals of* $\mathcal{L}$.
A formula Σ *is in class BR*(k)*-Quad if and only if*

1. Σ^{br} *belongs to Root, or*
2. *for the first maximum sub-clause* C' *of the first clause* $C \in \Sigma^{br}$ *for which* $(\Sigma \wedge \overline{C'})^{br}$ *is in class Root:*
   (a) *either* $(\Sigma \wedge \overline{C'})^{br}$ *is satisfiable, or*
   (b) *the formula* $(\Sigma \setminus \{C\}) \cup \{C'\}$ *is in class BR*(k)*-Quad.*

Now, let us show that $\bigcup$ *Quad* $\not\subset \bigcup$ *BR*(k)*-Quad* and $\bigcup$ *BR*(k)*-Quad* $\not\subset \bigcup$ *Quad*. Firstly, the following example shows that there exists Σ that does not belong to $\bigcup$ *Quad* but that belongs to $\bigcup$ *BR*(k)*-Quad*.

Example 32. *Let* $\Sigma =$
$\{\{a, b, c\}, \{\neg a, d, e\}, \{\neg d, \neg a, c\},$
$\{\neg e, d, i\}, \{\neg i, \neg a\}, \{\neg d, \neg e\}, \{a', b', c'\}, \{\neg a', d', e'\},$
$\{\neg d', \neg a', c'\}, \{\neg e', d', i'\}, \{\neg i', \neg a'\}, \{\neg d', \neg e'\}\}$.
After 3*-bounded resolution on* Σ*, we obtain* $\Sigma^{br} = \{\{\neg i, \neg a\}, \{\neg d, \neg e\}, \{\neg e, i\}, \{\neg a, d\}, \{\neg a, c\}, \{\neg e, \neg a\},$
$\{b, c\}, \{\neg i', \neg a'\}, \{\neg d', \neg e'\}, \{\neg e', i'\}, \{\neg a', d'\}, \{\neg a', c'\},$
$\{\neg e', \neg a'\}, \{b', c'\}\}$
All clauses of Σ^{br} *are binary, so* $\Sigma^{br} \in$ *Root and* $\Sigma \in \bigcup$ *BR*(3)*-Quad (for any order).*
Notice that Σ *contains two similar sets of clauses with two disjoint sets of variables. Hence for any ordering, there is no max-sub-clause* C' *such that* $(\Sigma \wedge \overline{C'})^*$ *is in class Root, because one of these two sets of clauses will remain unchanged and this set does not belong to Root and there does not exist any max-sub-clause such that* $(\Sigma \wedge \overline{C'})^*$ *is in Root. Consequently,* Σ *does not belong to* $\bigcup$ *Quad.*

The next example shows that there exists Σ that belongs to $\bigcup$ *Quad* but that does not belong to $\bigcup$ *BR*(k)*-Quad*.

Example 33. *Let* $\Sigma =$
$\{\{\neg w, \neg x, \neg y, \neg z\}, \{\neg d, \neg e, \neg f, \neg g\}, \{\neg a, b\}, \{a, b\}\}$
First, let us show that $\Sigma \notin \bigcup$ *BR*(k)*-Quad.*
For any value of $k \geq 1$*, we have that* $\Sigma^{br} = \{\{\neg w, \neg x, \neg y, \neg z\}, \{\neg d, \neg e, \neg f, \neg g\}, \{b\}\}$*, which does not belong to Root. Moreover, for any ordering of literals, there does not exist any max-sub-clause* C' *such that* $(\Sigma \wedge \overline{C'})^{br}$ *belongs to Root. Consequently* Σ *does not belong to* $\bigcup$ *BR*(k)*-Quad (for any k).*
It is easy to prove that for any ordering where b is the highest literal, Σ *belongs to* $\bigcup$ *Quad.* $\Sigma^* = \Sigma$ *is not in Root, therefore assume that the first max-sub-clause is* $\{\neg a\}$ *issued from* $\{\neg a, b\}$*. We have that* $(\Sigma \wedge (a))^*$ *has no positive clause and thus belongs to Root. Consequently,* Σ *belongs to* $\bigcup$ *Quad.*

These two last examples show that $\bigcup$ *Quad* and $\bigcup$ *BR*(k)*-Quad* are incomparable in the general case.
According to the complexity of bounded resolution, it is easy to show that:

Theorem 38. *The satisfiability and the membership of a CNF Σ to a given BR(k)-Quad fragment can be determined in $O(\max(d,k) \times (|\Sigma| + kn^k)^2)$ time, where n is the number of variables of Σ and d is the size of the longest clause in Σ.*

Proof. The complexity of checking the satisfiability and the membership to *BR(k)-Quad* follows from the complexity of both *Quad* and bounded resolution. Indeed, the complexity of both checking the satisfiability and the membership of a formula Σ to *Quad* is in $O(d \times |\Sigma|^2)$. As the maximum number of resolvents that can be generated by bounded resolution is n^k where n is the number of variables of Σ, the number of clauses of Σ can thus be increased with an additional n^k clauses of size k, leading to a formula of size $|\Sigma| + kn^k$. Taking into account this increase of the size of Σ and the complexity of the bounded resolution, the following worst case complexity is thus $O(\max(d,k) \times (|\Sigma| + kn^k)^2)$. □

Chapter 5

UP-based polynomial fragments of SAT

This chapter explores several polynomial fragments of SAT that are based on the unit propagation (UP) mechanism. In this chapter we consider other possible uses of UP to detect polynomial SAT instances. As a case study, we consider UP-based reductions of SAT instances into Horn instances [122, 123]. Noticeably, other polynomial classes could be selected as alternative target classes.
Actually, we combine several reduction processes. First, we compare two existing polynomial fragments based on UP: namely, Quad [6] and UP-Horn [10]. We answer an open question about the connections between these two classes: we show that UP-Horn and some other UP-based variants are strict subclasses of $\bigcup$Quad, where $\bigcup$Quad is the union of all Quad classes obtained by investigating all possible orderings of clauses.
Then, following [124], we exploit the very general paradigm consisting of the elimination of redundant clauses that can be detected thanks to UP and that, at the same time, do not belong to the targeted polynomial class.
Interestingly, many benchmarks from the DIMACS repository [11] and from SAT competitions [7] are shown to belong to UP-Horn. This result can be interpreted as a step towards the integration of theoretical investigations about tractable fragments within practical SAT solving.

5.1 UP-Horn (UP-reverse-Horn, UP-bin) vs. Quad

Firstly, we define UP-T clause and UP-T class where T is a tractable class.
Since our goal is to reduce UP-T instance to T instance by using the following approach:
We replace a clause C in UP-T instance by a clause C' in T instance (a T clause, see definition 205) where $C' \subseteq C$ and $\Sigma \vDash^* C'$.
So, we will focus on tractable classes that are closed under union (see definition 3.6.3).

Definition 205. *(T clause)*
Let T be a closed under union tractable class and $\Sigma \in$T be a CNF. Let C be a clause of Σ. C is called a T clause of Σ (or simply a T clause).

Definition 206. *(UP-T clause)*
Let T be a closed under union tractable class and $\Sigma \in$T be a CNF. Let C be a clause of Σ. C is called a UP-T clause of Σ if and only if there exists a T clause $C' \subseteq C$ s.t. $\Sigma \vDash^* C'$.

Definition 207. *(UP-T class)*
UP-T class is the class of CNFs that contain clauses that are T or UP-T.

We have the following examples: since Horn class, reverse-Horn class (see section 3.3.2), 2SAT class (see section 3.3.1) are closed under union tractable classes, so we have the following special cases of definitions 206 and 207.

Example 34. *(Some examples of UP-T classes).*

1. *(UP-Horn clause, UP-Horn class).*
   Let $C = \{\neg n_1, \ldots, \neg n_r, p_1, \ldots, p_s\}$, with $r \geq 0$ and $s \geq 1$, be a clause of Σ. C is called a UP-Horn clause of Σ if and only if there exists a Horn clause $C' \subseteq C$ s.t. $\Sigma \vDash^* C'$.
   UP-Horn class is the class of CNFs that contain clauses that are Horn or UP-Horn.

2. *(UP-reverse-Horn clause, UP-reverse-Horn class).*
   Let $C = \{\neg n_1, \ldots, \neg n_r, p_1, \ldots, p_s\}$, with $r \geq 1$ and $s \geq 0$, be a clause of Σ. C is called a UP-reverse-Horn clause of Σ if and only if there exists a reverse-Horn clause $C' \subseteq C$ s.t. $\Sigma \vDash^* C'$.
   UP-reverse-Horn class is the class of CNFs that contain clauses that are reverse-Horn or UP-reverse-Horn.

3. *(UP-bin clause, UP-bin class).*
   Let C be a clause of Σ. C is called a UP-bin clause of Σ if and only if there exists a binary clause $C' \subseteq C$ s.t. $\Sigma \vDash^* C'$.
   UP-bin class is the class of CNFs that contain clauses that are binary or UP-bin.

Remark 24. *We can also define UP-Tovey clause and UP-Tovey class (see section 6.2) in spite of Tovey class is not closed under union (it is just closed under disjoint union, see definition 175) but we can use our approach on Tovey class because the definition of Tovey class depends on the occurrences of the variables, and the occurrences of the variables will not increase by using our approach.*

We study these four examples of UP-T classes in the current and forthcoming chapters.

Let us now turn our attention to UP-Horn (resp., UP-reverse-Horn, UP-bin) vs. Quad. In [10], it is shown that these two SAT fragments are incomparable due to the fact that Quad depends on a selected ordering of clauses while UP-Horn (resp., UP-reverseHorn, UP-bin) does not depend on any ordering of clauses. In this section, we answer a remaining open question: "are UP-Horn (resp., UP-reverseHorn, UP-bin) and Quad equivalent, provided that Quad is considered on all possible orderings of clauses?". We show that UP-Horn (resp., UP-reverseHorn, UP-bin) is strictly included in $\bigcup$Quad.

Theorem 39.

A) UP-Horn $\subset \bigcup$Quad

B) UP-reverse-Horn $\subset \bigcup$Quad

C) UP-bin $\subset \bigcup$Quad.

Proof. First of all, let us prove the following lemma.

Property 74. *Let Σ be a CNF and C be a sub-clause of D s.t. $D \in \Sigma$ and $C \in \Sigma^*$.*

1. *If $\Sigma \wedge \neg(D \setminus C) \vDash^* \bot$ then $\bot \in \Sigma^*$.*

2. *If $\Sigma \wedge \neg(D \setminus C) \nvDash^* \bot$ and $F \subseteq D$ is a clause s.t. $\Sigma \wedge \neg F \vDash^* \bot$ then $\Sigma \wedge \neg(F \cap C) \vDash^* \bot$.*

Proof of Lemma 74.

1. Let $D = \{a_1, a_2, \ldots, a_n\}$ and $C = \{a_1, a_2, \ldots, a_r\}$ with $r \leqslant n$. $\Sigma \wedge \neg(D \setminus C) \vDash^* \bot$, can be written as follows:
$$\Sigma \wedge \bigwedge_{j \in \{r+1,\ldots,n\}} \{\neg a_j\} \vDash^* \bot \tag{5.1}$$
   As C is a sub-clause of D s.t. $D \in \Sigma$ and $C \in \Sigma^*$, C is then obtained from D by unit propagation on Σ. This means that all literals a_j with $j \in \{r+1, \ldots, n\}$ are already propagated from Σ. In consequence, from (5.1) we have $\Sigma \vDash^* \bot$, in other words $\bot \in \Sigma^*$.

2. $\Sigma \wedge \neg(D \setminus C) \nvDash^* \bot$ means that $\bot \notin \Sigma^*$ and for each sub-clause E of $(D \setminus C)$, $\Sigma \wedge \neg E \nvDash^* \bot$. If $F \subseteq D$ then $F \setminus (F \cap C) \subseteq (D \setminus C)$, therefore
$$\Sigma \wedge \neg(F \setminus (F \cap C)) \nvDash^* \bot \tag{5.2}$$

Assume that $\Sigma \wedge \neg(F \cap C) \not\models^* \bot$, with (5.2) we obtain that $\Sigma \wedge \neg F \not\models^* \bot$. Indeed Σ itself is able to unit propagate anything in $D \setminus C$ (since C was obtained by UP from D) and we do not derive any contradiction inside $D \setminus C$. Also, we do not derive any contradiction by UP from $\Sigma \wedge \neg(F \cap C)$ and so we cannot derive any contradiction by UP from $\Sigma \wedge \neg F$: this contradicts the initial hypothesis.

□

Now, let us prove the first assertion, namely A).
Assume that Σ is UP-Horn and $\Sigma \notin \bigcup$Quad. We assume also that Σ^* contains positive clauses, because otherwise $\Sigma^* \in$ Root and thus $\Sigma \in \bigcup$Quad. Let P a positive clause of Σ^*. Let M a clause of Σ s.t. $P \subseteq M$. P is obtained from M by unit propagation.

- If $\Sigma \wedge \neg(M \setminus P) \models^* \bot$ then $\bot \in \Sigma^*$ (by Lemma 74.1) and hence $\Sigma \in \bigcup$Quad.
- If $\Sigma \wedge \neg(M \setminus P) \not\models^* \bot$ then $\forall E \subseteq (M \setminus P), \Sigma \wedge \neg E \not\models^* \bot$. Σ is UP-Horn, so there exists a Horn clause H s.t. $H \subset M$ and $\Sigma \wedge \neg H \models^* \bot$. Therefore, $H \not\subseteq (M \setminus P)$ which means that $H \cap P \neq \emptyset$. H is a Horn clause and P is a positive one, in consequence $H \cap P = \{p\}$, i.e., $H \setminus \{p\} \subset (M \setminus P)$. By Lemma 74.2 we have: $\Sigma \wedge \{\neg p\} \models^* \bot$. So, for each positive clause P_i for $i = 1 \ldots n$, there is a variable p_i s.t.

$$\Sigma \wedge \{\neg p_i\} \models^* \bot \tag{5.3}$$

Now, let $C'_1, C'_2, \ldots, C'_{n'}$ be the clauses of Σ^* s.t. their negative literals are only formed by some of the $\neg p_i$'s. As previously, let D_i be a clause of Σ s.t. $C'_i \subseteq D_i$ ($i \in \{1, \ldots, n'\}$). Once again, we assume that $\Sigma \wedge \neg(D_i \setminus C'_i) \not\models^* \bot$ because Lemma 74.1 would directly entail that Σ belongs to Root otherwise. So, we have:

$$\forall E' \subseteq (D_i \setminus C'_i), \Sigma \wedge \neg E' \not\models^* \bot \tag{5.4}$$

And we can apply the same reasoning: Σ is UP-Horn, thus $\exists H' \subset D_i$ s.t. H' is Horn and $\Sigma \wedge \neg H' \models^* \bot$ and since $H' \not\subseteq D_i \setminus C'_i$, we also have $H' \cap C'_i \neq \emptyset$. As H' is a Horn clause, two cases are to be distinguished (note that $(H' \setminus (H' \cap C'_i)) \subseteq D_i$ and consequently $(H' \setminus (H' \cap C'_i)) \subset D_i \setminus C'_i$):

1. $H' \cap C'_i = \{\neg p_{i_1}, \neg p_{i_2}, \ldots, \neg p_{i_m}\}$ has no positive literal. Since $\Sigma \wedge \neg H' \models^* \bot$ and (5.4), we have $\Sigma \wedge \neg\{\neg p_{i_1}, \neg p_{i_2}, \ldots, \neg p_{i_m}\} \models^* \bot$, i.e., $\Sigma \wedge (p_{i_1}) \wedge (p_{i_2}) \wedge \cdots \wedge (p_{i_m}) \models^* \bot$ which means that there exists a sub-clause $\{\neg p_{i_1}, \neg p_{i_2}, \ldots, \neg p_{i_m}\}$ of Σ s.t. $(\Sigma \wedge \neg\{\neg p_{i_1}, \neg p_{i_2}, \ldots, \neg p_{i_m}\})^*$ contains an empty clause and so belongs to Root, which entails that $\Sigma \in \bigcup$Quad.
2. $H' \cap C'_i = \{p'_j, \neg p_{i_1}, \neg p_{i_2}, \ldots, \neg p_{i_m}\}$ contains one positive literal. Thanks to (5.3), (5.4) and $\Sigma \wedge \neg H' \models^* \bot$, we have:

$$\Sigma \wedge \{\neg p'_j\} \models^* \bot \tag{5.5}$$

Similarly, we can perform the same reasoning with $C''_1, C''_2, \ldots, C''_{n''}$ the clauses of Σ^* s.t. their negative literals are only formed by some of $\neg p_i$ and $\neg p'_j$, we obtain $\Sigma \wedge \{\neg p''_k\} \models^* \bot$, and so on.

Now, if we take the total order s.t. all p_i precede all p'_j which precede all p''_k and so on and all of these literals precede all the other ones of Σ (i.e., $p_1 < p_2 < \cdots < p_n < p'_1 < \cdots < p'_{n'} < p''_1 < \cdots < p''_{n''} < \cdots < p'^k_1 < \cdots < p'^k_{n'^k} < \ldots$), we can apply Dalal's procedure and it is possible to remove all p_i, all p'_j, all p''_k and so on. Thus the clauses that contain these positive literals are also removed. The same reasoning is applied until the resulting formula does not contain any positive clause; this latter formula belongs to Root, and consequently Σ is in $\bigcup$Quad.

The second assertion, namely B), can be proved in a similar way.

The last assertion about UP-bin can be proved as follows.
Let Σ a CNF in UP-bin. For each non-binary clause $C = \{p_1, p_2, \ldots, p_n\}$ of Σ, we have $\Sigma \models^* \{p_i, p_j\}$ with $\{i, j\} \subseteq \{1, \ldots, n\}$ (because $\Sigma \in$ UP-bin) which means $\Sigma \models (\Sigma \setminus \{C\}) \cup \{p_i, p_j\}$, i.e., Σ can be replaced

by $(\Sigma \setminus \{C\}) \cup \{p_i, p_j\}$. When this transformation is iterated for each non-binary clause, we obtain a binary formula, which is in Root, and thus Σ is in $\bigcup$Quad.

To show that the inclusion is strict, we give the following example.

Example 35. *Consider the following CNF:* $\Sigma = \{\{a, b, c\}, \{\neg d, \neg e, f\}\}$. *It is easy to show that* Σ *belongs to Quad but does neither belong to UP-Horn, nor UP-reverse-Horn nor UP-bin.*

First, Σ *belongs to Quad. Indeed, since* $\Sigma^* = \Sigma$ *does not contain any negative clause,* $\Sigma^* \in$ *Root and thus* $\Sigma \in$ *Quad.* Σ *is not in UP-Horn because* $(\Sigma \wedge \neg a)^* = \{\{b, c\}, \{\neg d, \neg e, f\}\} \nvDash^* \bot$ *and similar results hold for* $(\Sigma \wedge \neg b)^*$ *and* $(\Sigma \wedge \neg c)^*$. Σ *is not in UP-reverse-Horn because none of* $\{\neg d\}, \{\neg e\}, \{f\}, \{\neg d, f\}$ *and* $\{\neg e, f\}$ *follows from* Σ *through* $\vdash^*$. Σ *is not in U-bin. Indeed, no binary sub-clauses of* $\{a, b, c\}$ *and* $\{\neg d, \neg e, f\}$ *can be derived from* Σ *according to* $\vdash^*$.

□

5.2 Extending the range of UP-based reductions

As it is well-known UP-redundant clauses (see definition 83) can be safely removed from Σ [124]. Let us now add two properties, leading to two additional reduction operators, namely *UP-NRes* and *UP-PRes*, respectively.

Property 75. *Let* $C = \{\neg n_1, \ldots, \neg n_n, p_1, \ldots, p_p\}$ *be a clause of* Σ.
If $\Sigma \models^* \{\neg n_1, \ldots, \neg n_n\}$ *or*
$(\exists\ p_i \in C$ *s.t.* $\Sigma \models^* \{\neg n_1, \ldots, \neg n_n, p_i\}$ *and* $\Sigma \models^* \{\neg n_1, \ldots, \neg n_n, \neg p_i\})$,
then $\Sigma \models \{\neg n_1, \ldots, \neg n_n\}$.

Definition 208 (UP-NRes(C))**.** *When a clause* $C = \{\neg n_1, \ldots, \neg n_n, p_1, \ldots, p_p\}$ *of* Σ *satisfies Property 75, UP-NRes*(C) *is defined as* $\{\neg n_1, \ldots, \neg n_n\}$.

Property 76. *Let* $C = \{\neg n_1, \ldots, \neg n_n, p_1, \ldots, p_p\}$ *be a clause of* Σ.
If $\exists\ p_i \in C$ *s.t.* $\Sigma \not\models^* \{\neg n_1, \ldots, \neg n_n, p_i\}$ *and* $\Sigma \models^* \{\neg n_1, \ldots, \neg n_n, \neg p_i\}$,
then $\Sigma \models^* \{\neg n_1, \ldots, \neg n_n, p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_p\}$.

Definition 209 (UP-PRes(C))**.** *When a clause* $C = \{\neg n_1, \ldots, \neg n_n, p_1, \ldots, p_p\}$ *of* Σ *satisfies Property 76 with respect to the literals* p_i *to* p_j, *UP-PRes*(C) *is defined as* $\{\neg n_1, \ldots, \neg n_n, p_1, \ldots, p_{i-1}, p_{j+1}, \ldots, p_p\}$.

The following remark shows how the previous properties are used to recognize SAT instances that can be reduced in polynomial time into Horn instances.

Remark 25. *Algorithm 17 uses the previous properties to recognize SAT instances that can be reduced in polynomial time into Horn instances.*
It works as follows.

1. *All Horn clauses recorded in* Σ'.
2. *All remaining clauses C are checked one by one.*
3. *By Property 75, when the negative part of C is UP-derivable from* Σ, *this negative part is a new Horn clause and so it is recorded in* Σ' .
4. *Otherwise, the second part of Property 75 is checked in lines 8 and 9.*
5. *While Property 76 is verified in lines 8 to 11.*
6. *To obtain UP-PRes*(C), *the tests are down in lines 12 to 14. It inserts the smallest clause (with respect to its number of positive literals) within* Σ'.
7. *The function* `RedundancyUP` *is called in line 15 to get rid of UP-redundant clauses as described in [124].*
8. *By using the function isHorn, the initial formula* Σ *is reduced into Horn formula (line 16).*

Algorithm 17: isHorn-by-extendedUP

Input: a CNF Σ
Output: *true* if Σ is UP-Horn; *false* otherwise

$\Sigma' \leftarrow \{C | C \in \Sigma$ s.t. `isHorn`$(\{C\})\}$;
foreach $C \in \Sigma$ *s.t.* $C = \{\neg n_1, \ldots, \neg n_n, p_1, \ldots, p_p\}$, $n \geq 0$ *and* $p > 1$ **do**
  if $(\Sigma \models^* \{\neg n_1, \ldots, \neg n_n\})$ **then** $\Sigma' \leftarrow \Sigma' \cup \{\{\neg n_1, \ldots, \neg n_n\}\}$;
  else
    $\Sigma'' \leftarrow \emptyset$;
    $C' \leftarrow C$;
    forall the $p_i \in C$ **do**
      if $(\Sigma \models^* \{\neg n_1, \ldots, \neg n_n, p_i\})$ **then**
        if $(\Sigma \models^* \{\neg n_1, \ldots, \neg n_n, \neg p_i\})$ **then** $\Sigma' \leftarrow \Sigma' \cup \{\{\neg n_1, \ldots, \neg n_n\}\}$;
        else $\Sigma'' \leftarrow \Sigma'' \cup \{\{\neg n_1, \ldots, \neg n_n, p_i\}\}$;
      else if $(\Sigma \models^* \{\neg n_1, \ldots, \neg n_n, \neg p_i\})$ **then** $C' \leftarrow C' \setminus \{p_i\}$;
    if $(\{\neg n_1, \ldots, \neg n_n\} \notin \Sigma')$ **then**
      if $(\Sigma'' = \emptyset)$ **then** $\Sigma' \leftarrow \Sigma' \cup \{C'\}$;
      else $\Sigma' \leftarrow \Sigma' \cup \Sigma''$;
$\Sigma' \leftarrow$ `RedundancyUP`(Σ') ;
return `isHorn` (Σ');

Function RedundancyUP

Input: a CNF Σ
Output: an UP-irredundant SAT instance Γ equivalent to Σ with respect to satisfiability

$\Gamma \longleftarrow \Sigma$;
foreach $C \in \Sigma$ *sorted according to their decreasing sizes* **do**
  if $(\Gamma \setminus C \vDash^* C)$ **then** $\Gamma \longleftarrow \Gamma \setminus \{C\}$;
return Γ;

Function isHorn

Input: a CNF Σ
Output: *TRUE* if Σ is a Horn formula; *FALSE* otherwise

foreach $C \in \Sigma$ **do**
  if *(it exists more than one positive literal in C)* **then return** FALSE ;
return TRUE ;

Implementation and experimentation

To give the practical interest of algorithm 17, a variant of algorithm 17 is implemented. The implementation has the following aspects:

1. The reduced CNF depends on the order according to which the clauses are considered.
2. The program is iterated until no new Horn clause is produced (to perform all possible simplifications) and it runs on various benchmarks from the DIMACS repository [11] and from the SAT competitions (`www.satcompetition.org`).
3. All experimentations have been conducted on an Intel(R) Xeon(TM) CPU 3.00 GHz with 2GB of memory under Linux CentOS release 4.1.

We get the following interesting results:

1. Some instances were reduced to polynomial-time ones, running the program just once. In Table 5.1, all instances that could be reduced to instances belonging to the Horn class are given: 99 instances reduced to instances of the Horn class have been found within (almost) 1600 tested instances. For each instance, its name, its size (#var. and #cla.), the number of propagations (#UP) and the time spent in seconds to reduce the instance to UP-Horn are given. When the program is iterated until no new Horn clause is produced, 28 additional instances are reduced to Horn. They are given in Table 5.2 where "removed cla" (resp. "removed var") represents the ratio (in percents) of clauses (resp. variables) removed by the method and where "#lit" represents the total number of literals that have been removed.

2. Even when this pre-treatment does not conduct the instance to be reduced into a polynomial-time one, the global size of the instance is often decreased in a significant manner, whereas its polynomial subpart is increased accordingly.

To give its interest as pre-treatment, a combination of the pre-treatment with Minisat are compared. We get the following results:

1. In Table 5.3, the time required to solve instances using Minisat [125] with the time spent by a combination of the pre-treatment with Minisat are compared. In this table, the columns "Minisat" represent the time consumed by Minisat to solve the original instance ("original") and the simplified one ("simplified"); and the columns "%profit" represents the gain (in percents) obtained by the pre-treatment when the simplification time is taken into account either together with the satisfiability checking time ("total") or not ("partial").

2. As examples, we have the following:
   Minisat required 204.54 seconds to solve `IBM_05_SAT_dat.k100`. The simplification of this instance by means of our process took 133.17 seconds: it removed more than 40000 variables (21% of the total number of variables) and more than 282000 clauses (27% of the total number of clauses). This simplified benchmark was solved by Minisat in 28.02 seconds, only. If we also take the time spent for simplification into account, the global time to simplify and solve the benchmark is thus 161.19 seconds, to be compared with the 240.54 seconds needed by Minisat to solve the initial instance.

3. In general, the time spent to simplify the instance is largely compensated by the saved time in the solving step. Most often, simplifying SAT instances by means of our pre-processing allows computing time to be saved, globally.

4. In rare situations, the simplified benchmark is more time-consuming to solve than the original one (e.g., `ip50`) or the spent time to simplify is not compensated during the subsequent solving step (e.g., `fifo8_100`).

CNF instances	# var.	# cla.	#UP	time (s.)	CNF instances	# var.	# cla.	#UP	time (s.)
aim-100-1_6-yes1-4	100	160	179	0	IBM_FV_2004_rule_batch...				
aim-100-2_0-yes1-2	100	200	456	0	IBM_...04_SAT_dat.k15	15300	65598	397812	0.25
aim-100-6_0-yes1-1	100	600	2502	0	IBM_...05_SAT_dat.k15	25128	134922	1708357	1.22
aim-100-6_0-yes1-2	100	600	2534	0	IBM_...15_SAT_dat.k100	226970	893496	2432156	2.46
aim-100-6_0-yes1-3	100	600	777	0	IBM_...15_SAT_dat.k15	30790	119911	184301	0.19
aim-100-6_0-yes1-4	100	600	568	0	IBM_...15_SAT_dat.k20	42330	165416	252596	0.26
aim-200-6_0-yes1-2	200	1200	6113	0.01	IBM_...15_SAT_dat.k25	53870	210921	329216	0.33
aim-200-6_0-yes1-4	200	1200	696	0	IBM_...15_SAT_dat.k30	65410	256426	413391	0.42
aim-50-2_0-yes1-2	50	100	218	0	IBM_...15_SAT_dat.k35	76950	301931	506031	0.5
aim-50-2_0-yes1-3	50	100	250	0	IBM_...15_SAT_dat.k40	88490	347436	606086	0.6
aim-50-2_0-yes1-4	50	100	156	0	IBM_...15_SAT_dat.k45	100030	392941	714746	0.71
aim-50-6_0-yes1-1	50	300	516	0	IBM_...15_SAT_dat.k50	111570	438446	830681	0.83
aim-50-6_0-yes1-2	50	300	692	0	IBM_...15_SAT_dat.k55	123110	483951	955361	0.99
aim-50-6_0-yes1-3	50	300	440	0	IBM_...15_SAT_dat.k60	134650	529456	1087176	1.07
aim-50-6_0-yes1-4	50	300	1621	0	IBM_...15_SAT_dat.k65	146190	574961	1227876	1.22
cnf-r1-b3-k1.2	660004	5281	56944	0.21	IBM_...15_SAT_dat.k70	157730	620466	1375571	1.38
cnf-r1-b4-k1.1	397893	7089	105048	0.18	IBM_...15_SAT_dat.k75	169270	665971	1532291	1.53
cnf-r1-b4-k1.2	922148	6818	60079	0.29	IBM_...15_SAT_dat.k80	180810	711476	1695866	1.69
cnf-r2-b2-k1.2	406052	6064	54402	0.15	IBM_...15_SAT_dat.k85	192350	756981	1868606	1.88
cnf-r2-b3-k1.2	668180	9169	100807	0.27	IBM_...15_SAT_dat.k90	203890	802486	2048061	2.06
cnf-r2-b4-k1.1	406052	12784	178182	0.25	IBM_...15_SAT_dat.k95	215430	847991	2236821	2.26
cnf-r2-b4-k1.2	930282	12464	175575	0.37	IBM_...22_SAT_dat.k10	18919	77414	596987	0.4
jnh10	100	850	6737	0.02	IBM_...22_SAT_dat.k15	29833	122814	1249118	0.96
jnh11	100	850	11187	0.02	IBM_...22_SAT_dat.k20	40753	168249	1845706	1.48
jnh12	100	850	5323	0.01	iso-brn005.shuffled	1130	9866	13572	0.02
jnh13	100	850	4940	0.01	f19-b21-s0-0	746	3517	23805	0.03
jnh14	100	850	3362	0.01	f27-b10-s0-0	193	1113	8268	0.01
jnh15	100	850	7544	0.01	f27-b1-s0-0	193	1113	9401	0.01
jnh18	100	850	16943	0.03	f27-b2-s0-0	193	1113	5614	0.01
jnh19	100	850	10836	0.02	f27-b3-s0-0	193	1113	8716	0.01
jnh202	100	800	4641	0.01	f27-b4-s0-0	193	1113	5992	0.01
jnh203	100	800	18563	0.03	f27-b5-s0-0	193	1113	5626	0.01
jnh208	100	800	16108	0.03	f27-b8-s0-0	193	1113	7702	0.01
jnh20	100	850	8478	0.02	f27-b9-s0-0	193	1113	8684	0.01
jnh211	100	800	3030	0.01	f83-b11-s0-0	1000	43900	318968	0.74
jnh214	100	800	12131	0.02	f83-b14-s0-0	1000	43540	811348	1.61
jnh215	100	800	10558	0.02	f83-b17-s0-0	1000	43900	180456	0.37
jnh216	100	800	12821	0.02	par8-1-c	64	254	5613	0
jnh2	100	850	2201	0	par8-1	350	1149	9224	0
jnh302	100	900	246	0	par8-2	350	1157	7641	0
jnh303	100	900	13452	0.03	par8-4-c	67	266	6216	0
jnh304	100	900	1720	0	par8-4	350	1155	10248	0.01
jnh305	100	900	5348	0.01	par8-5	350	1171	7978	0
jnh307	100	900	2211	0	pitch.boehm	1192	6361	656	0.01
jnh308	100	900	15155	0.03	qg5-10.shuffled	1000	43900	318968	0.69
jnh309	100	900	2460	0.01	qg6- 10.shuffled	1000	43540	811348	1.62
jnh310	100	900	3054	0.01	qg7-10.shuffled	1000	43900	180456	0.37
jnh4	100	850	5955	0.01	3col20_5_5.shuffled	40	176	774	0
jnh5	100	850	4151	0.01	3col20_5_6.shuffled	40	176	656	0
jnh8	100	850	4749	0.01	3col20_5_7.shuffled	40	176	903	0
jnh9	100	850	3099	0.01	3col20_5_9.shuffled	40	176	438	0

Table 5.1: UP-Horn instances

	instance size		removed				
CNF Instances	#var.	#cla.	cla	var	#lit.	#UP	time (s.)
een-tipb-sr06-par1	163647	484831	94%	95%	252004	68362283	38.98
ezfact16_10.shuffled	193	1113	26%	34%	335	5614	0.01
ezfact16_3.shuffled	193	1113	37%	44%	479	5992	0.01
f32-b2-s0-0	40	176	70%	69%	178	941	0
f32-b4-s0-0	40	176	85%	77%	163	919	0
f33-b9-s0-0	80	346	88%	80%	391	5867	0
f6-b2-s2-20	478	1007	95%	92%	532	14216	0
IBM_FV_2004_rule_batch_03_SAT_dat.k30	29079	118925	44%	55%	31075	1393665	1.07
IBM_FV_2004_rule_batch_05_SAT_dat.k10	15399	81447	87%	93%	33203	1252239	0.76
IBM_FV_2004_rule_batch_05_SAT_dat.k20	34863	188452	74%	82%	72024	7798669	5.49
IBM_FV_2004_rule_batch_05_SAT_dat.k25	44598	241982	67%	75%	86760	18851484	16.38
IBM_FV_2004_rule_batch_05_SAT_dat.k30	54333	295512	60%	67%	99477	31503131	26.84
IBM_FV_2004_rule_batch_06_SAT_dat.k15	17501	75616	43%	49%	18130	1278040	1.07
IBM_FV_2004_rule_batch_06_SAT_dat.k20	23826	103226	71%	78%	41764	12961178	10.17
IBM_FV_2004_rule_batch_10_SAT_dat.k15	40278	159501	33%	35%	26022	8285670	6.89
IBM_FV_2004_rule_batch_1_11_SAT_dat.k10	28280	111519	47%	49%	25573	58410957	42.46
IBM_FV_2004_rule_batch_18_SAT_dat.k10	17141	69989	48%	55%	19878	13050828	8.7
IBM_FV_2004_rule_batch_19_SAT_dat.k10	21823	83902	24%	31%	13250	298260	0.26
IBM_FV_2004_rule_batch_19_SAT_dat.k15	34697	134023	17%	22%	14917	508638	0.47
IBM_FV_2004_rule_batch_19_SAT_dat.k20	47577	184178	17%	23%	23258	14607263	12.98
IBM_FV_2004_rule_batch_20_SAT_dat.k10	17567	72087	36%	41%	14004	5226452	3.63
IBM_FV_2004_rule_batch_21_SAT_dat.k10	15919	65180	35%	39%	11897	267966	0.21
IBM_FV_2004_rule_batch_21_SAT_dat.k15	25213	103881	25%	28%	13564	471438	0.39
IBM_FV_2004_rule_batch_21_SAT_dat.k20	34513	142616	26%	30%	21454	9624852	7.38
IBM_FV_2004_rule_batch_22_SAT_dat.k25	51673	213684	24%	27%	28739	30219471	22.32
IBM_FV_2004_rule_batch_23_SAT_dat.k10	18612	76086	41%	48%	16035	69713	0.09
IBM_FV_2004_rule_batch_27_SAT_dat.k10	6477	27070	62%	70%	10054	3826810	2.15
rip08.boehm	471	263	92%	59%	145	8728	0.01
x6dn.boehm	521	1255	86%	84%	1022	137818	0.07

Table 5.2: Reduction of SAT instances using several runs

	Minisat		% profit		instance size		removed				
CNF Instance	Original	Simplified	partial	total	#var	#cla	var	cla	#lit	#UP	time (s.)
f2clk_40	293.4	265.19	9	7	27568	80439	36%	36%	13619	5009951	5.52
f3-b29-s0-10	76.72	26.58	65	65	2125	12677	24%	35%	3520	127851	0.18
f28-b4-s0-0	3.15	0.04	98	96	769	4777	13%	22%	927	45554	0.08
f81-b3-s0-0	2081.34	1654.61	20	17	33385	163232	26%	30%	24855	36519004	63.69
fifo8_100	14.31	11.12	22	-48	64762	176313	42%	46%	42718	8435460	9.98
fifo8_200	43.74	77.5	-78	-134	129762	353513	37%	40%	76361	18309357	24.51
fifo8_300	349.92	152.11	56	45	194762	530713	35%	39%	109878	28532439	39.94
fifo8_400	500.73	428.59	14	3	259762	707913	34%	38%	143413	38349604	55.85
IBM_03_SAT_dat.k60	28.33	11.99	57	17	59649	244535	22%	27%	33386	12915029	11.33
IBM_03_SAT_dat.k90	195.45	173.44	11	1	90219	370145	17%	20%	38507	21481064	18.32
IBM_05_SAT_dat.k100	204.54	28.02	86	21	190623	1044932	21%	27%	167316	143847825	133.17
IBM_05_SAT_dat.k60	55.59	10.52	81	-37	112743	616692	31%	37%	123076	73459545	65.46
IBM_16_1_SAT_dat.k95	14.62	2.18	85	29	50492	203817	23%	26%	24509	9440470	8.16
ip50	92.63	307.54	-233	-266	66131	214786	36%	44%	47569	23195373	30.61
logistics-rotate-09t6	80.07	6.5	91	-55	8186	887558	15%	30%	908	157186029	117.23

Table 5.3: Some typical instances

Chapter 6

Extensions of Tovey's polynomial fragment of SAT

A first result in this chapter is a family of extensions [126, 123] of one well-known Tovey's polynomial fragment [9] so that they also include instances that can be simplified using UP. The second result is a comparison between $\bigcup$Quad and G-UP-Tovey.
The chapter is organized as follows. In the next section 6.1, we present the work of Tovey and the related works of it, then in section 6.2, we extend Tovey's class in several directions using UP. In section 6.3 we show that $\bigcup$Quad and G-UP-Tovey are incomparable.

6.1 The occurrence of variables and Tovey classes

Tovey in [9] studied the effect of the number of occurrence of variables per clauses on the NP-completeness of SAT problem. That is, if we let (k, s)-SAT denote the SAT problem where each clause has exactly k distinct literals and each variable occurs at most s times then Tovey shows that (3,4)-SAT remains NP-complete and shows that (3,3)-SAT is trivial (always satisfiable). He also shows that the SAT instances where each variable occurs at most twice is solved in linear time.

We present here the results of Tovey's paper and some of related results that have been presented by other authors [127, 128, 129, 130, 131, 132].

6.1.1 Tovey's work

In this section we present the results of Tovey [9].
Let (k, s)-SAT=$\{\Sigma \in \text{SAT}: \forall C \in C(\Sigma), |C| = k, \forall l_1, l_2 \in C, V(l_1) \neq V(l_2), \forall v \in V(\Sigma), Occ_\Sigma(v) \leq s\}$.
Tovey first reduces 3-SAT to the following instances.

Theorem 40. *[9] .*
Boolean satisfiability is NP-complete when restricted to instances with 2 or 3 variables per clause and at most 3 occurrences per variable.

Corollary 7. *[9].*
For any $s \geq 3$, either

1. *every Boolean expression with exactly 3 variables per clause and no more than s occurrences per variable, is satisfiable, or*
2. *3, s-SAT is NP-complete..*

Tovey also proved the following.

Theorem 41. *($(3, 4)$-SAT)[9].*
$(3, 4)$-SAT is NP-complete.

If the number of occurrences of all variables is less than or equal the length of the clauses then,

Theorem 42. *(r, r-SAT) [9].*
Every instance of r, r-SAT is satisfiable.

The 2-occurrence case

From corollary 7, theorem 41, and theorem 42 all the cases of (k, s)-SAT where $s \geq 3$ are solved (it is either satisfiable or NP-complete). The only remaining case is when $s = 2$ that is every variable appears at most 2 times.
For this case, Tovey presents a polynomial time algorithm, in fact it is linear time algorithm since (in modern terminology) he used the two linear-time techniques that are used in modern SAT-solver to simplify the given CNF formula, **the UP-technique** (to remove repeatedly the unit clauses form the given CNF to get a new CNF without unit clauses) and **monotone-literal technique** (to assign the value TRUE to the literals that appears monotony [appears just positively or just negatively]).
After applying these two techniques, he gets an equivalent CNF (if there is an empty clause during the UP procedure where in this case the SAT instance is unsatisfiable) that satisfies the following two conditions.

1. All the clauses of the result CNF have length greater than one.
2. Each variable appears once complemented and once uncomplemented.

But if a CNF satisfies these two conditions then it is satisfiable according to the following lemma.

Property 77. *A given CNF is satisfiable if it satisfies the two conditions 1 and 2.*

Also Tovey [9] conjectured the following statement.

Conjecture 3. *If $s \leq 2^{r-1} - 1$, then every instance of r, s-SAT is satisfiable.*

But Dubois in [130] disproved this conjecture, see the next section.

6.1.2 The works related to Tovey's result

Dubois' work

First, Dubois [130] disproved the conjecture of Tovey: if $s \leq 2^{r-1} - 1$, then every instance of r, s-SAT is satisfiable.
We note that if $r < 4$ and $s \leq 2^{r-1} - 1$ then the instances r, s-SAT are satisfiable, if $r = 1$, $s = 1$ or $r = 2$ and hence $s = 1$ then the instances r, s-SAT are trivially satisfiable and if $r = 3$ and hence $s = 3$ then the instances r, s-SAT are satisfiable according to theorem 42.
Therefore to disprove the conjecture one have to find an unsatisfiable instance r, s-SAT with $r \geq 4$ and $s \leq 2^{r-1} - 1$.
If $r = 4$ then $s \leq 7$ but Dubois constructed $4, 6$-SAT unsatisfiable instance and hence disproved Tovey conjecture.
Recursively, Dubois constructed 1-SAT, 2-SAT, 3-SAT,... unsatisfiable instances for looking for the smallest possible values of s.
For that, two transformations are introduced that preserve the satisfiability and they were made to move from an unsatisfiable r-SAT instance to an unsatisfiable $r + 1$-SAT instance.

Now we turn to the other related works by Dubois in [130]. First he proved that if all instances of a given class r, s-SAT with exactly s occurrences per variable are satisfiable, then all instances of the class of instances r, s-SAT with at most s occurrences per variable are satisfiable, by proving that if there is an unsatisfiable instance r, s-SAT with at most s occurrences per variable then one can construct from it another unsatisfiable instance r, s-SAT with exactly s occurrences per variable.

Proposition 2. *[130]*
If all instances of a given class r, s-SAT with exactly s occurrences per variable are satisfiable the class of instances r, s-SAT is satisfiable.

Using proposition 2, Dubois [130] proved the following proposition where $[x]$ denotes the integral part of x, i.e., the greatest integer less than or equal x.

Proposition 3. *Let r_0, s_0-SAT be a satisfiable class of instances; then the class of instances $r_0 + 1, s_0 + [\frac{s_0}{r_0}]$-SAT is satisfiable.*

Thanks to proposition 3, Dubois [130] proved the following theorem.

Theorem 43. *Let r_0, s_0-SAT be a satisfiable class of instances. Then any class of instances r, s-SAT such that $r = r_0 + \lambda$ and $s \leq s_0 + \lambda[\frac{s_0}{r_0}]$ with $\lambda \in N$, is satisfiable.*

Proof. Apply proposition 3 λ times and use the fact $[\frac{s_0+\lambda[\frac{s_0}{r_0}]}{r_0+\lambda}] = [\frac{s_0}{r_0}], \forall \lambda \in N$, we have that the class $r_0 + \lambda, s_0 + \lambda[\frac{s_0}{r_0}]$-SAT is satisfiable. □

Finally, using theorem 43 Dubois [130] reproved theorem 42 of Tovey.

Corollary 8. *Every instance of r, r-SAT is satisfiable.*

Proof. The class $1, 1$-SAT is trivially satisfiable, so all classes r, r-SAT are satisfiable. □

The work by Benhamou

Belaid Benhamou in [133] obtained the same result of corollary 8 that the class $1, 1$-SAT is trivially satisfiable, so all classes r, r-SAT are satisfiable, he proves that by using a relationship between matching in bipartite graphs and the satisfiability problem, he gives an algorithm which finds a model for such formulas in polynomial time complexity if one exists or, if not, proves in polynomial time complexity that the current formula is not an element of the restricted class.

The work by Berman, Karpinski and Scott

Tovey (see section 6.1.1) showed that 3-SAT remains NP-complete if each variable occurs at most 4 times but 3-SAT is always satisfiable if each variable occurs at most 3 times. Berman, Karpinski and Scott in [134] studied the relation between these two problems by showing the following:

1. Let $k \geq 0$ be a fixed integer, the 3-SAT instances in which k variables occur four times and the remaining variables occur three times has a polynomial time satisfiability algorithm.
2. The satisfiability of 3-SAT instances in which all but one variable occur three times, and the remaining variable occurs an arbitrary number of times, is NP-complete.

In this section we present this work.

Notation 17. *[134]*

1. *Let $(3, 4^{(k)})$-SAT denotes the set of 3-SAT instances in which k variables occur four times and the remaining variables occur three times.*
2. *Let $(3, 4^{(k)}, n)$-SAT denotes the set of instances of $(3, 4^{(k)})$-SAT with n variables.*

According to this notation we have the following proposition.

Proposition 4. *[135]*

1. *Every instance of $(3, 4^{(3)})$-SAT is satisfiable.*
2. *There are unsatisfiable instances of $(3, 4^{(9)})$-SAT.*

The main theorem in this work is the following.

Theorem 44. *[134]*
The satisfiability of instances of $(3, 4^{(k)}, n)$-SAT can be determined in time $2^{k/3} n^{k/3} poly(n)$.

The proof of theorem 44 used the following definition.‘

Definition 210. *(Witness function for a satisfying assignment) [134]*
Let I be a satisfiable CNF and ϕ be a satisfying assignment for I. A witness function $w : C(I) \to V(I)$ for ϕ is a function such that, $\forall C \in C(I), \exists L(W(C)) \in C$ and $\phi(L(W(C)) = TRUE$.

So, $w(C)$ is a variable that refers to the " witnesses " of the satisfaction of C in ϕ.
To prove theorem 44 the authors use the following lemma:

Property 78. *[134]*
If I is a satisfiable instance of $(3, 4^{(k)})$-SAT then there is a satisfying assignment ϕ with a surjective witness function $w : C(I) \to V(I)$ for ϕ.

Hence, by theorem 44, for any fixed k, $(3, 4^{(k)}, n)$-SAT instances can be solved in polynomial time.

Then the authors in [134] proved that if the occurrence of at least one variable in 3-SAT is arbitrary then it remains NP-complete.

Theorem 45. *The restriction of 3-SAT to the set of instances in which all but one variable occur exactly three times is NP-complete.*

We give a little modified proof.

Proof. We give a reduction from 3-SAT similar to one given by Tovey.

1. We can assume without loss of generality that every variable appears at least twice (because if there are variables appearing once, we can assign the value TRUE to the literals associated with these variables and remove the clauses containing these variables and repeat this process until we get an equivalent 3-SAT with every variable appearing at least twice).

2. If a variable x occurs exactly three times we do nothing. If x occurs $d \neq 3$ times (i.e., $d = 2$ or $d > 3$), we introduce new variables $x_1, \ldots, x_d$ and add the clauses $x_i \vee \neg x_{i+1}, i = 1, \ldots, d-1$ and the clause $x_d \vee \neg x_1$. Then we replace the d occurrences of x by $x_1, \ldots, x_d$.
   Hence, we get an equivalent SAT instance I with clauses of lengths 2 and 3, and every variable appears exactly three times.

3. Take two copies I_1, I_2 of I (with disjoint sets of variables) and a new variable x. Add x to every clause of lengths 2 in I_1 and $\neg x$ to every clause of lengths 2 in I_2. The resulting instance is equivalent to I and every variable except one in it appears exactly three times.

□

6.2 Extensions of one Tovey's fragment

From now on, we call " Tovey's class" the class of CNF where any variable occurs at most twice (counting occurrences of both positive and negative literals).[1]

Definition 211 (Tovey CNF)**.** .
Let Σ be a CNF, $\Sigma \in$ Tovey if and only if every variable appearing in Σ occurs at most twice in Σ.

It is well-known that the (un)satisfiability of a CNF Σ is preserved when a clause C of Σ is replaced by one of its sub-clauses C', provided that $\Sigma \models C'$.

Property 79. *Let Σ be a CNF, C be a clause of Σ and C' be such that $C' \subseteq C$. When $\Sigma \models C'$ we have that Σ is satisfiable if and only if $(\Sigma \setminus \{C\}) \cup \{C'\}$ is satisfiable.*

[1] Actually, Tovey defined other polynomial fragments, too, see section 6.1.1

The class of CNF that can be reduced into Tovey's instances thanks to this property is defined as follows. We call it $\bigcup$Tovey.

Definition 212 ($\bigcup$Tovey CNF)**.** .
Let $\Sigma = \{C_1, \dots C_m\}$ be a CNF, Σ belongs to $\bigcup$Tovey if and only if $\exists\ \Gamma = \{C'_1, \dots\ C'_m\} \in$ Tovey s.t. $\forall 1 \leq i \leq m, C'_i \subseteq C_i$ and $\Sigma \vDash C'_i$.

Remark 26. *Unfortunately, checking the membership to $\bigcup$Tovey is intractable.*
To circumvent this issue, we apply Property 79 in specific circumstances to find polynomial-time recognizable and solvable extensions of Tovey; specifically, we apply the property only when $\Sigma \vDash^* C'$ so that we can benefit from the linear-time complexity of UP.
Hence, we look for sub-classes of $\bigcup$Tovey for which instances can be recognized in polynomial time.

A concept of a clause that is Tovey in a CNF will prove useful in the following.

Definition 213 (Tovey clause)**.** .
A clause C is a Tovey clause with respect to Σ if and only if every variable occurring in C occurs at most twice in Σ.

The UP-Tovey clause can be defined in the following.

Definition 214 (UP-Tovey clause)**.** .
A clause $C \in \Sigma$ is called a UP-Tovey clause if and only if $\exists\ C' \subseteq C$ s.t. C' is a Tovey clause with respect to Σ and $\Sigma \vDash^* C'$.

Let us now define a unit-propagation-based Tovey class. We call it UP-Tovey.

Definition 215 (UP-Tovey CNF)**.** .
A CNF formula $\Sigma \in$ UP-Tovey if and only if every clause of Σ is a UP-Tovey clause.

Obviously, checking the satisfiability of a CNF belonging to UP-Tovey can be achieved in polynomial time.
The following well-known property will prove useful to provide a polynomial algorithm that checks whether or not a given CNF belongs to UP-Tovey,

Property 80. *Let Σ be a CNF and C be a clause. If $\Sigma \nvDash^* C$ then $\forall C' \subset C$, $\Sigma \nvDash^* C'$.*

Accordingly, when a clause C is not UP-Tovey in Σ, it is sufficient to check whether the longest sub-clause C' that is Tovey is such that $\Sigma \vDash^* C'$ or not.

Definition 216. *(Maximum Tovey sub-clause).*
Let Σ be a CNF and $C \in \Sigma$. A clause C' s.t. $C' \subset C$ is the maximum Tovey sub-clause of C in Σ if and only if C' is a Tovey clause in Σ and $\nexists\ C''$ s.t. that $C' \subset C'' \subset C$ and C'' is a Tovey clause in Σ.

Property 81. *Let Σ be a CNF, $\Sigma \in$ UP-Tovey if and only if every clause C in Σ that is not Tovey in Σ contains a sub-clause C' that is a maximum Tovey sub-clause of C in Σ and such that $\Sigma \vDash^* C'$.*

Thus, to check if a given CNF belongs to UP-Tovey, we only need to check for each clause of Σ if its unique maximum Tovey sub-clause is a UP-consequence of Σ. Consequently, checking the membership of a CNF to UP-Tovey and checking the satisfiability of a CNF belonging to UP-Tovey can both be achieved in polynomial time.

Example 36. *Let $\Sigma = \{\{a, b, c\}, \{\neg a, b\}, \{\neg a, c\}\}$ be a CNF. It is easy to check that the following CNF belongs to UP-Tovey.*
Indeed, $\{b, c\}$ is the maximum Tovey sub-clause of $\{a, b, c\}$ and $\Sigma \vDash^* \{b, c\}$; as $\{\{b, c\}, \{\neg a, b\}, \{\neg a, c\}\}$ belongs to Tovey, we have that Σ belongs UP-Tovey.

Clearly, we have that

Algorithm 18: Check-membership-in-UP-Tovey

Input: a CNF Σ
Output: *TRUE* if $\Sigma \in$ UP-Tovey, *FALSE* otherwise

```
1 foreach C ∈ Σ do
2     C' ← maxToveySubClause(C, Σ);
3     if (C' = ∅) or (Σ ⊭* C') then return FALSE ;
4 return TRUE ;
```

Function maxToveySubClause

Input: a clause C and a CNF Σ s.t. $C \in \Sigma$
Output: the maximum Tovey sub-clause of C with respect to Σ

```
1 C' ← ∅ ;
2 foreach l ∈ C do
3     Let v_l the boolean variable issued from the literal l ;
4     if (v_l occurs at most twice in Σ) then C' ← C' ∪ {l} ;
5 return C' ;
```

Property 82. *UP-Tovey* $\subset \bigcup$*Tovey.*

Remark 27. *A recognition algorithm for checking the membership of* Σ *in UP-Tovey needs thus to verify, for each clause* $C \in \Sigma$ *that is not Tovey in* Σ*, whether or not the maximum Tovey sub-clause of* C *in* Σ *can be deduced from* Σ *by unit propagation.*

Algorithm 18 does the job, clauses of Σ *are processed one by one according to a static ordering on the clauses of* Σ*.*
Note that any such ordering can be used: this does not influence the result. C'*, the maximum Tovey sub-clause of* C *in* Σ*, is extracted by the function* `maxToveySubClause`*. If such a clause is not empty and can be deduced from* Σ *by UP then the next clause is processed. In the other case,* C *is not UP-Tovey in* Σ *and the algorithm returns* FALSE*.*

In the worst-case, each clause is processed by unit propagation only once: the worst case time complexity of the algorithm is thus in $O(m \times |\Sigma|)$, where m is the number of clauses in the CNF.

Property 83. *Let* Σ *be a CNF of* m *clauses. The worst-case complexity of Algorithm 18 is in* $O(m \times |\Sigma|)$*.*

Algorithm 18 is complete for UP-Tovey.

Property 84. *Let* Σ *be a CNF,* $\Sigma \in$ *UP-Tovey if and only if Check-membership-in-UP-Tovey*(Σ) *returns TRUE.*

Remark 28. *Interestingly, it is possible to improve Algorithm 18 in such a way that it attempts to recognize also (some) CNF that do not belong to UP-Tovey but that however belong to* $\bigcup$*Tovey, while keeping a polynomial time worst-case complexity. It will allow the reduced instance* Γ *to contain clauses that are UP-Tovey in* Γ *but that are not UP-Tovey in* Σ*. We call such a class UP*$^+$*-Tovey and thus make sure that UP-Tovey*$\subseteq$ *UP*$^+$*-Tovey* $\subset \bigcup$*Tovey.*

To this end, we take advantage of the following properties.

- *Maximum Tovey sub-clauses can be checked in the currently reduced instance* Γ *under construction (instead of " in the initial* Σ*").*
- *There is no loss of relevant information by checking* $\Gamma \vDash^* C'$ *instead of checking* $\Sigma \vDash^* C'$*.*

Algorithm 19: Check-membership-in-UP$^+$-Tovey

Input: a CNF Σ
Output: *TRUE* if a proof of $\Sigma \in$ UP$^+$-Tovey is found; *FALSE* otherwise

```
1 Γ ← Reduction(Σ) ;
2 return isTovey(Γ);
```

- *Without altering the worst-case complexity, when we have $\Gamma \not\models^* C'$, instead of ending the procedure, we can still process the other C clauses in Γ. The goal is to deliver a CNF instance with shortened clauses, so that the whole process can be iterated on this instance (and subsequent ones) with some possible additional shortenings being done at each iteration step.*

Algorithm 19 implements these ideas. First, Function Reduction is called to tentatively shorten clauses of Σ. Then, it simply verifies whether the resulting CNF is Tovey (line 2) by calling the function `isTovey`*. Note that in* `Reduction` *all instructions are achieved on Σ itself; thus Σ plays also the role of the CNF under construction. All clauses of Σ are processed one by one in the* `Reduction` *function.*

According to a static ordering of the clauses of Σ, the function does not stop when one clause C that is not UP-Tovey in the current Σ has been discovered. It processes the remaining clauses in Σ. The hope is that C will actually become a Tovey clause in Σ at the end of the whole (iterated) reduction process. Obviously, Algorithm 19 remains incomplete for $\bigcup$UP-Tovey; especially, contrary to Algorithm 18, the order according to which clauses are processed can influence the success of the reduction.

Moreover, when this algorithm returns FALSE*, some of the clauses of Σ might have been substituted by sub-clauses. Accordingly, Algorithm 19 can be iterated in order to recognize even more $\bigcup$Tovey instances.*

In the sequel, we present a generalization of the UP-Tovey class by defining a hierarchy of classes, called G-UP-Tovey$_<(i)$ where $<$ is a given total ordering of clauses ("G" standing for "Generalized"). For convenience, we will drop the $<$ parameter that is indexing the class.

Definition 217 (G-UP-Tovey(i) CNF)**.** .
Let Σ be a CNF made of m clauses. The sequence $\Sigma_1, \Sigma_2, \ldots, \Sigma_m$ is defined as follows.

- $\Sigma_1 = Reduction(\Sigma)$
- $\Sigma_{i+1} = Reduction(\Sigma_i)$*, where* $1 \leq i < m$.

$\forall i \in [1..m] : \Sigma \in$ *G-UP-Tovey*(i) *if and only if Σ_i is Tovey.*

Remark 29. *Clearly, whenever $\Sigma_{i+1} = \Sigma_i$, no further change can occur in Σ_j where $j \geq i$. Since each call to* `Reduction` *makes at least one clause become Tovey in the CNF under construction, it is guaranteed that applying* `Reduction` *on Σ_m would not change Σ_m when m is the number of clauses in Σ.*

Alternatively, we could have defined $\Sigma_{i+1} = (\Sigma_i \backslash C) \cup D$ for all $C \in \Sigma_i$ s.t. $\exists\, D \subset C$ a maximum Tovey sub-clause in Σ_i and $\Sigma_i \models^* D$.

Algorithm 20 depicts a way to determine for a given CNF Σ, whether there exists a lowest i such that Σ belongs to G-UP-Tovey(i) or not. In the positive case this lowest index i is found and is the output of the Algorithm.

Notes 37. *It is easy to show that G-UP-Tovey$(i) \subset$ G-UP-Tovey$(i+1)$ for $i \geq 1$. Indeed, any CNF formula Σ that belongs to G-UP-Tovey(i) also belongs to G-UP-Tovey(j) for $j > i$, whereas the converse is not TRUE.*

Property 85. *Let Σ be a CNF made of m clauses. The complexity of Algorithm 20 is in $O(m^2 \times |\Sigma|)$.*

Proof. As the number of calls to Algorithm 20 to `Reduction` is bound by m, the overall complexity is then in $O(m^2 \times |\Sigma|)$. □

Function Reduction

Input: a CNF Σ

Output: a tentative reduction of Σ into a Tovey CNF

1 **foreach** $C \in \Sigma$ **do**
2 $\quad C' \leftarrow$ `maxToveySubClause`(C, Σ);
3 $\quad$ **if** $(C' \neq \emptyset)$ and $(\Sigma \vDash^* C')$ **then** $\Sigma \leftarrow (\Sigma \backslash \{C\}) \cup \{C'\}$;
4 **return** Σ ;

Function isTovey

Input: a CNF Σ

Output: *TRUE* if Σ is a Tovey *FALSE* otherwise

1 **foreach** *v boolean variable occurring in* Σ **do**
2 $\quad$ **if** *(v occurs more than twice in* Σ*)* **then return** FALSE ;
3 **return** TRUE ;

Clearly, we have:

Property 86. *Let $<$ be a total ordering of clauses.*
UP-Tovey = G-UP-Tovey$(1) \subset$ *G-UP-Tovey*$(2) \subset ... \subset$ *G-UP-Tovey*$(m-1) \subset$ *G-UP-Tovey*$(m) \subset \bigcup$*Tovey.*

Checking the membership to G-UP-Tovey(i) is in $O(m^2 \times |\Sigma|)$.

Example 37. *The formula $\Sigma = \{\{a,b,c\},\{\neg c,d,e,h\},\{\neg c,f\},\{\neg d,\neg e\},\{\neg f,h\},\{\neg f,\neg h\}\}$ shows that the above hierarchy does not collapse since it belongs to G-UP-Tovey*(2) *and not to UP-Tovey (just consider the order of clauses as they appear in the formula).*

Firstly, $\{\neg f,h\}$ is not a Tovey clause in Σ and has no Tovey sub-clause in Σ. Consequently, Σ does neither belong to Tovey nor to UP-Tovey.

Secondly, $\{a,b\}$ is the maximum Tovey sub-clause of $\{a,b,c\}$ and $\Sigma \vDash^* \{a,b\}$. The other clauses in Σ are either Tovey in Σ or do not have any Tovey sub-clause in Σ. Hence, $\Sigma_1 = \{\{a,b\},\{\neg c,d,e,h\},\{\neg c,f\},\{\neg d,\neg e\},\{\neg f,h\},\{\neg f,\neg h\}\}$.

Now, consider Σ_1. We have that $\{\neg c,d,e\}$ is the maximum Tovey sub-clause of $\{\neg c,d,e,h\}$ and $\Sigma_1 \vDash^* \{\neg c,d,e\}$. We also have both that $\{\neg c\}$ is the maximum Tovey sub-clause of $\{\neg c,f\}$ and $\Sigma_1 \vDash^* \{\neg c\}$. The other clauses in Σ_1 are either Tovey in Σ_1 or do not have any Tovey sub-clause in Σ_1. Hence, $\Sigma_2 = \{\{a,b\},\{\neg c,d,e\},\{\neg c\},\{\neg d,\neg e\},\{\neg f,h\},\{\neg f,\neg h\}\}$. As Σ_2 belongs to Tovey, we have that Σ belongs to G-UP-Tovey(2).

Let us end this section by three remarks.

Remark 30.

1. *First, the G-UP-Tovey(i) tractable classes depend on the static ordering $<$ of the clauses. In the general case, to get rid of this dependence, we would need to consider all the possible orderings and the resulting recognition algorithm would then become exponential.*

2. *A second remark is even in the case where a static clauses ordering is considered, if we look for the smallest Tovey sub-clauses or simply the smallest sub-clause (line 3 of Algorithm Reduction) in the reduction phase, the recognition algorithm becomes exponential in the size of the longest clause.*

3. *Finally it must be noted that neither the UP-Tovey nor the G-UP-Tovey classes attempt to substitute a clause by a sub-clause that would contain variables occurring more than twice, although it might appear that other shortened clauses would decrease the total number of occurrences of these variables, making the reduced CNF become Tovey. This is another reason that conducts the proposed classes to be strict sub-classes of $\bigcup$Tovey.*

Algorithm 20: Membership-to-G-UP-Tovey(i)

Input: a CNF Σ made of m clauses

Output: The lowest i s.t. $\Sigma \in$ UP-Tovey(i); 0 if such i does not exist

```
1 Σ_0 ← Σ;
2 i ← 0;
3 while (i < m) do
4 |   Σ_{i+1} ← Reduction(Σ_i);
5 |   if ( isTovey(Σ_{i+1}) ) then return i + 1;
6 |   else i ← i + 1;
7 return 0;
```

6.3 Comparison between ⋃Quad and G-UP-Tovey

To complete the comparison between the various tractable classes considered in this thesis, let us end with the following results showing that ⋃Quad and G-UP-Tovey are incomparable in the general case.

Theorem 46. *For all $i \geq 1$,*

- *$\bigcup$Quad $\not\subseteq$ G-UP-Tovey(i)*
- *G-UP-Tovey(i) $\not\subseteq \bigcup$Quad.*

Proof. Let us build two counter-examples showing that $\subseteq$ does not hold. First, consider $\Sigma = \{\{a, \neg b\}, \{a, \neg c\}, \{a, \neg d\}\}$. Σ is in Quad since it does not contain any positive clause. However, Σ does neither belong to UP-Tovey, nor to UP-Tovey(i). Indeed, no sub-clause of any clause of Σ can be deduced from Σ. Now, consider $\Psi = \{\{a, b, c\}, \{d, e, f\}, \{\neg a, \neg b, \neg c\}\}$. Clearly, Ψ belongs to Tovey and thus to G-UP-Tovey(i). However, Ψ does not belong to Root and no sub-clause can be derived according to any ordering of clauses. Consequently, Ψ is not in $\bigcup$Quad. □

Part V

Conclusions and perspectives

Tractable classes of SAT (namely, the classes that can be solved and recognized in polynomial time or just solved in polynomial time when the recognition algorithm is not necessary (for example, for SLUR class that does not use the structural properties of CNFs but it uses a nondeterministic algorithm, a CNF in SLUR class can be solved in polynomial time without needing to know the recognition algorithm for the SLUR class) are interesting in their own right and could also play an increasing role in future efficient SAT solvers.

One can attempt to extend these classes while keeping polynomial solving and recognition algorithms. There are many techniques that can be used to that end. Some of them are used to obtain hierarchies of tractable classes such as the hierarchy of Gallo and Scutella [1], the hierarchy of Dalal and Etherington [2], the hierarchy of Pretolani [3], the hierarchies of Cepek and Kucera [4], the hierarchy of Kullmann [5], the hierarchy of Andrei et al. ($Rank_k$ hierarchy), and the hierarchies that generalize the SLUR class such as the $SLUR(i)$ hierarchy[84], the $SLUR^*(i)$ hierarchy [106] and the $SLUR_i$ hierarchy [107, 108].

In this thesis, the focus was on fragments of the clausal Boolean language for which instances are recognizable in polynomial time and satisfiability can be checked in polynomial time, too. Especially, we have investigated several possible roles of the unit propagation (UP) mechanism in this respect. Note that many of the well-known tractable classes (such as Horn [67, 68, 69, 2, 70, 71], reverse-Horn [73], renamable-Horn [73], extended Horn [76], hidden extended Horn[76], simple extended Horn [78], CC-balanced formulas [80], SLUR algorithm formulas [82, 83], almost-Horn [85], $F - Horn^*$ [86], ordered[87], ordered-renamable [87], almost ordered formulas [87], q-Horn [79, 82, 88, 89], matched [82], LinAut formulas[92, 25, 93], PURL algorithm [95], and the classes that depend on the occurrence of variables such as Tovey classes [9] and the classes that depend on the the ordering of clauses such as Quad classes [6]), already use in some way the linear-time UP procedure, which is also a basic operation in all DPLL-like procedures in SAT solvers.

Two main techniques have been used in this thesis: the first one is by using the linear-time UP procedure to entail a clause that subsumes at least one of the clauses of the given CNF. See [10] for a similar method that extends Horn, reverse-Horn and 2SAT classes to UP-Horn, UP-reverse-Horn and UP-bin, respectively. As a case study, a class called UP-Tovey has been defined whose instances can be recognized and solved in quadratic time, extending the Tovey classes [9] (the ones that depend on the occurrences of variables). The second technique is by replacing the incomplete UP procedure by another polynomial time procedure. In this respect, the focus was about Dalal's *Quad* fragments [6], which have attracted little attention so far. Bounded by an almost quadratic complexity, these fragments capture several well-known polynomial fragments like the Horn and binary ones and go beyond those fragments. We have studied properties of Dalal's fragments that can be used to extend these fragments. Especially, we have exhibited one extension of *Quad* that keeps its complexity intact, and exhibited polynomial variants. To this end, bounded resolution, which is an incomplete but polynomial time procedure [118], was used to extend *Quad*. Also, we have extended *Quad* by using different orders among clauses and subclauses instead of one unique order that was used in [6] among clauses and subclauses and we have shown that UP-Horn and other UP-based fragments are included in $\bigcup$Quad.

In the future, we plan to investigate to which extent the detection of those fragments can be grafted to existing SAT-solvers. We believe that progress in defining larger-cardinality polynomial fragments and in better understanding their limits and properties could help in the design of even more powerful SAT-solvers, which could detect these polynomial fragments, either during a pre-processing step and during the main search itself. Extensive experimentations should help us to assess the extent to which checking some of the polynomial fragments as a pre-processing step can improve SAT solvers. It should also help us to assess which fragments prove more helpful experimentally. Finally, a key issue that we plan to address in this respect is the definition and experimentation of heuristics that select (some of) the ordering(s) of clauses on which *Quad* and its variants are based, as considering all orderings is obviously out of reach, also trying of extend *Quad* to a class that does not depend on order (orders) for some NP-complete subclasses of SAT such 3SAT.

Although this solves open questions about the relationships between these fragments, UP-Horn remains of high interest since it does not depend on any ordering of clauses whereas each such ordering

gives rise to a specific *Quad* class. Noticeably, although G-UP-Tovey(i) and $\bigcup$Quad are based on the same inference rule (i.e., entailment modulo unit propagation), their reduction processes are different. Accordingly, another promising path for further research would consist in investigating how these processes could be mixed and hybridized, giving rise to variant tractable classes. One such path for further research is relaxing some constraints of *Quad* while remaining polynomial. Especially, the reduction to the *Root* class requires the UP-simplification by the negation of one single clause at a time. However for some instances, *Root* can only be reached when such a UP-simplification is done for several clauses together and when this leads to a satisfiable instance. This is a first easy extension of *Quad*.

Bibliography

[1] Giorgio Gallo and Maria Grazia Scutellà. Polynomially solvable satisfiability problems. *Information Processing Letters*, 29:221–227, 1988. (Cited pages 2, 66, 67, 68, 69, 72 and 121.)

[2] Mukesh Dalal and David W. Etherington. A hierarchy of tractable satisfiability problems. *Information Processing Letters*, 44(4):173–180, 1992. (Cited pages 2, 43, 66, 70, 71 and 121.)

[3] Daniele Pretolani. Hierarchies of polynomially solvable satisfiability problems. *Annals of Mathematics and Artificial Intelligence*, 17(2):339–357, 1996. (Cited pages 2, 66, 71, 72, 73, 75 and 121.)

[4] Ondrej Cepek and Petr Kucera. Known and new classes of generalized Horn formulae with polynomial recognition and SAT testing. *Discrete Applied Mathematics*, 149(1-3):14–52, 2005. (Cited pages 2, 66, 73, 74, 75, 84, 85 and 121.)

[5] Oliver Kullmann. Investigating a general hierarchy of polynomially decidable classes of CNF's based on short tree-like resolution proofs. Technical report, TR99-041, Electronic Colloquium on Computational Complexity (ECCC), 1999. (Cited pages 2, 66, 75, 76, 77, 78 and 121.)

[6] Mukesh Dalal. An almost quadratic class of satisfiability problems. In *Proceedings of the Twelfth European Conference on Artificial Intelligence (ECAI'96)*, pages 355–359. John Wiley, 1996. (Cited pages 2, 7, 89, 90, 91, 92, 93, 96, 101 and 121.)

[7] SAT-Competition. http://www.satcompetition.org/, 2013. (Cited pages 7 and 101.)

[8] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pages 151–158, 1971. (Cited pages 7 and 21.)

[9] Craig A. Tovey. A simplified NP-complete satisfiability problem. In *Discrete Applied Mathematics*, volume 8, pages 85–89, 1984. (Cited pages 7, 95, 109, 110 and 121.)

[10] Olivier Fourdrinoy, Éric Grégoire, Bertrand Mazure, and Lakhdar Saïs. Reducing hard SAT instances to polynomial ones. In *Proceedings of the IEEE International Conference on Information Reuse and Integration (IRI 2007)*, pages 18–23, 2007. (Cited pages 7, 101, 102 and 121.)

[11] DIMACS. Second challenge on satisfiability testing http://dimacs.rutgers.edu/Challenges/, 1993. (Cited pages 7, 101 and 105.)

[12] G. Boole. Les lois de la pensee. *Mathesis*, 1854. (Cited page 11.)

[13] Chin-Liang Chang and Richard Char-Tung Lee. Symbolic logic and mechanical theorem proving. *Academic Press*, 1973. (Cited page 11.)

[14] B. Mazure. *De la satisfaisabilité à la compilation de bases de connaissances propositionnelles*. PhD thesis, Université d'Artois, 1999. (Cited page 11.)

[15] D.A.Plaisted and S.Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computation*, 2:293–304, 1986. (Cited page 15.)

[16] Pierre Siegel. *Représentation et Utilisation de la connaissance en calcul propositionnel*. Thése d'état, Université de Provence, GIA–Luminy, Marseille (France), 1987. (Cited page 15.)

[17] Michael Sipser. *Introduction to the theory of computation*. THOMSON COURSE TECHNOLOGY, second edition, 2006. (Cited page 19.)

[18] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, New York, NY, USA, 1st edition, 2009. (Cited page 19.)

[19] John C. Martin. *Introduction to Languages and The Theory of Computation*. The Mc Graw Hill Companies, 4th edition, 2011. (Cited pages 19 and 20.)

[20] Steven Homer and Alan Selman. *computability and complexity theory*. Springer-Verlag New York, 2001. (Cited page 19.)

[21] Christos H. Papadimitriou. *Computational Complexity*. Addison Wesley Pub. Co., 1994. (Cited page 19.)

[22] John C. Martin. *Introduction to Languages and The Theory of Computation*. The Mc Graw Hill Companies, 1st edition, 1991. (Cited page 19.)

[23] P. van Beek F. Rossi and T. Walsh. *Handbook of Constraint Programming*. 2006. (Cited page 21.)

[24] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the Association for Computing Machinery*, 7:201–215, 1960. (Cited pages 21, 22 and 42.)

[25] Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, February 2009. (Cited pages 21, 22, 26, 59 and 121.)

[26] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem proving. *Journal of the Association for Computing Machinery*, 5:394–397, 1962. (Cited page 22.)

[27] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, pages 530–535, 2001. (Cited pages 22 and 27.)

[28] João P. Marques Silva and Karem A. Sakallah. Grasp - a new search algorithm for satisfiability. In *ICCAD*, pages 220–227, 1996. (Cited pages 22 and 26.)

[29] Chu Min Li and Anbulagan Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proceedings of the 15th international joint conference on Artifical intelligence - Volume 1*, pages 366–371, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc. (Cited page 22.)

[30] Niklas Eén and Niklas Sörenson. An extensible sat-solver. In *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*, volume 2929 of *Lecture Notes in Computer Science*, pages 333–336, Santa Margherita Ligure, Italy, May 2003. Published in 2004. Springer. (Cited page 22.)

[31] Niklas Sörensson and Niklas Eén. Minisat 2.1 and minisat++ 1.0 - sat race 2008 editions. *SAT 2009 competitive events booklet: preliminary version*, page 31, 2009. (Cited page 22.)

[32] Geoffrey Chu and Peter J. Stuckey. Pminisat : A parallelization of minisat 2.0. Solver description, sat-race 2008, 2008. (Cited page 22.)

[33] Luís Gil, Paulo Flores, and Luís M. Silveira. PMSat: a parallel version of MiniSAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 6:71–98, 2008. (Cited page 22.)

[34] Youssef Hamadi, Saïd Jabbour, and Lakhdar Saïs. What we can learn from conflicts in propositional satisfiability. *Annals of Operations Research*, pages 1–25, 2015. (Cited pages 23, 24, 25, 26, 27, 35 and 36.)

[35] Lakhdar Sais. Probleme SAT:progres et defis: in french language. *London*, 2008. (Cited page 26.)

[36] C. P. Gomes, B. Selman, N. Crato, and H. Kautz. Heavy-tail phenomena in satisfiability and constraint satisfaction. *Journal of Automated Reasoning*, 24(1/2):67–100, 2000. (Cited page 26.)

[37] J. Lynce, I.and Marques-Silva. Efficient data structures for backtrack search SAT solvers. *Annals of Mathematics and Artificial Intelligence*, page 43, 2005. (Cited page 26.)

[38] Tanbir Ahmed. An implementation of the DPLL algorithm. Thesis (m.s.), Concordia University, 2009. (Cited pages 26, 28 and 42.)

[39] Coudert O. On solving covering problems. In *Proceedings of the ACM/IEEE Design Automation Conference*, page 197–202, 1996. (Cited page 27.)

[40] Zhang H. and Stickel M. Implementing the davis-putnam method. In *Proceedings of SAT 2000*, page 309–326. eds. I. Gent, H. van Maaren and T. Walsh, 2000. (Cited page 27.)

[41] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: engineering an efficient sat solver. In *DAC '01: Proceedings of the 38th annual Design Automation Conference*, pages 530–535, New York, NY, USA, June 2001. ACM. (Cited page 27.)

[42] Roberto J. Bayardo Jr. and Robert C. Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI'97)*, pages 203–208, Providence (Rhode Island, USA), July 1997. (Cited page 27.)

[43] Jon William Freeman. *Improvements to Propositional Satisfiability Search Algorithms*. Ph.d. thesis, University of Pennsylvania, Department of Computer and Information Science, 1995. (Cited page 27.)

[44] John N. Hooker and V. Vinay. Branching rules for satisfiability (extended abstract). In *Proceedings of the 14th Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 426–437, London, UK, 1994. Springer-Verlag. (Cited page 27.)

[45] Robert G. Jeroslow and Jinchang Wang. Solving propositional satisfiability problems. *Annals of Mathematics and Artificial Intelligence*, 1:167–187, 1990. (Cited page 27.)

[46] Hantao Zhang. Sato: An efficient prepositional prover. In William McCune, editor, *Automated Deduction–CADE-14*, volume 1249 of *Lecture Notes in Computer Science*, pages 272–275. Springer Berlin / Heidelberg, 1997. (Cited page 27.)

[47] M. Ouyang. *Implementation of the DPLL algorithm*. PhD thesis, Rutgers University, 1999. (Cited page 27.)

[48] Niklas Eén and Armin Biere. Effective preprocessing in SAT through variable and clause elimination. In *Proceedings of the Eighth International Conference on Theory and Applications of Satisfiability Testing (SAT'05)*, volume 3569 of *Lecture Notes in Computer Science*, pages 61–75, St. Andrews, Scottland, June 2005. Springer. (Cited pages 28, 31 and 32.)

[49] Sathiamoorthy Subbarayan and Dhiraj K. Pradhan. NiVER: Non-increasing variable elimination resolution for preprocessing SAT instances. In *Proceedings of the Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT'04)*, pages 276–291, 2004. (Cited pages 28, 30 and 31.)

[50] Fahiem Bacchus and Jonathan Winter. Effective preprocessing with hyper-resolution and equality reduction. In *Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT-2003)*, pages 341–355, 2003. (Cited pages 28, 29 and 30.)

[51] Fahiem Bacchus. Enhancing davis putnam with extended binary clause reasoning. In *National Conference on Artificial Intelligence*, pages 613–619, 2002. (Cited page 28.)

[52] Van Gelder A. and Tsuji Y. K. *Satisfiability testing with more reasoning and less guessing*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, page 559–586. 1996. (Cited page 28.)

[53] J. Franco. Elimination of infrequent variables improves average case performance of satisfiability algorithms. *SIAM Journal on Computing*, 20:1119–1127, 1991. (Cited pages 28 and 30.)

[54] A. Van Gelder. Combining preorder and postorder resolution in a satisfiability solver. *Kautz, H.,and Selman, B., eds., Electronic Notes of SAT*, 2001. (Cited pages 28 and 30.)

[55] R. I. Brafman. A simplifier for propositional formulas with many binary clauses. *IJCAI*, pages 515–522, 2001. (Cited pages 28 and 30.)

[56] R. Ostrowski, É. Grégoire, B. Mazure, and L. Saïs. Recovering and exploiting structural knowledge from CNF formulas. In *Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming(CP'02)*, volume 2470 of *Lecture Notes in Computer Science*, pages 185–199. Springer, 2002. (Cited pages 28, 32 and 33.)

[57] Jerome Lang and Pierre Marquis. Complexity results for independence and definability in propositional logic. *In Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98)*, page 356– 367, 1998. (Cited page 33.)

[58] Oliver Kullmann. Worst-caseanalysis,3-SAT decision and lower bounds:approaches for improved sat algorithms. *DIMACS Proceedings SAT Workshop, DIMACS Series in Discrete Mathematics and Theorical Computer Science, American Mathematical Society*, 1996. (Cited page 33.)

[59] C. Piette, Y. Hamadi, and L. Sais. Vivifying propositional clausal formulae. *In proceedings of the 18th European Conference on Artificial Intelligence (ECAI'2008)*, 2008. (Cited pages 34 and 35.)

[60] N.Een and N.Sorensson. Minisat-a sat solver with conflict-clause minimization. *In Proceedings of the Eighth International Conference on Theory and Applications of Satisfiability Testing (SAT'05)*, 2005. (Cited page 36.)

[61] T. J. Schaefer. The complexity of satisfiability problems. *In Proceedings 10th Symposium on Theory of Computing,ACM Press*, page 216–226, 1978. (Cited pages 39 and 64.)

[62] Robert Endre Tarjan. Depth first search and linear graph algorithms. *SIAM Journal of Computing*, 1:146–160, 1972. (Cited page 40.)

[63] W. Victor Marek. Introduction to mathematics of satisfiability. *Chapman and Hall/CRC*, 2009. (Cited page 41.)

[64] Krom Melven R. The decision problem for a class of first-order formulas in which all disjunctions are binary. *Mathematik*, 13:15–20, 1967. (Cited page 41.)

[65] Bengt Aspvall, Michael F. Plass, and Robert E. Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, 8(3):121–123, 1979. (Cited pages 41 and 42.)

[66] S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM Journal of Computing*, 5:691–703, 1976. (Cited page 41.)

[67] Wiliam F. Dowling and Jean H. Gallier. Linear-time algorithms for testing satisfiability of propositional Horn formulae. *Journal of Logic Programming*, pages 267–284, 1984. (Cited pages 43 and 121.)

[68] Maria Grazia Scutellá. A note on Dowling and Gallier's top-down algorithm for propositional Horn satisfiability. *Journal of Logic Programming*, 8(3):265–273, 1990. (Cited pages 43 and 121.)

[69] Malik Ghallab and Gonzalo Escalada-Imaz. A linear control algorithm for a class of rule-based systems. *Journal of Logic Programming*, 11(2):117 – 132, 1991. (Cited pages 43 and 121.)

[70] Antoine Rauzy. Polynomial restrictions of SAT: What can be done with an efficient implementation of the Davis and Putnam's procedure? In *Proceedings of the First International Conference on Principles and Practice of Constraint Programming (CP '95)*, volume 976 of *Lecture Notes in Computer Science*, pages 515–532. Springer, 1995. (Cited pages 43 and 121.)

[71] Michel Minoux. LTUR: a simplified linear-time unit resolution algorithm for Horn formulae and computer implementation. *Information Processing Letters*, 29(1):1–12, 1988. (Cited pages 43 and 121.)

[72] H. R. Lewis. Renaming a set of clauses as a Horn set. *Journal of the ACM*, 25:134–135, 1978. (Cited pages 43 and 44.)

[73] Bengt Aspvall. Recognizing disguised NR(1) instances of the satisfiability problem. *J. Algorithms*, 1(1):97–103, 1980. (Cited pages 43, 44 and 121.)

[74] Chandru V., Coullard Collette R., Peter L. Hammer, M. Montanuz, and Xiaorong Sun. On renamable Horn and generalized Horn functions. *Ann. Math. Artif. Intell.*, 1:33–47, 1990. (Cited pages 43 and 72.)

[75] Jean-Jacques Hébrard. A linear algorithm for renaming a set of clauses as a Horn set. *Theor. Comput. Sci.*, 124(2):343–350, 1994. (Cited pages 43 and 44.)

[76] V. Chandru and J. N. Hooker. Extended Horn sets in propositional logic. *Journal of the ACM*, 38(1):205–221, 1991. (Cited pages 45, 46 and 121.)

[77] John Franco and Sean Weaver. *Handbook of Combinatorial Optimization*, chapter Algorithms for the Satisfiability Problem, pages 311–455. Springer Science+Business Media, 2013. (Cited pages 45, 51, 56, 57, 66 and 84.)

[78] R.P. Swaminathan and D.K. Wagner. The arborescence–realization problem. *Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science*, 59(3):267–283, 1995. (Cited pages 46 and 121.)

[79] John Franco. Relative size of certain polynomial time solvable subclasses of satisfiability. *American Mathematical Society*, DIMACS Series on Discrete Mathematics and Theoretical Computer Science 35::211–224, 1997. (Cited pages 46, 55, 56 and 121.)

[80] Michele Conforti, Gérard Cornuéjols, and Kristina Vuskovic. Balanced matrices. *Discrete Mathematics*, 306(19-20):2411–2437, 2006. (Cited pages 46, 47, 48, 49 and 121.)

[81] G. Zambelli. A polynomial recognition algorithm for balanced matrices. *J. Combin. Theory B 95*, page 49–67, 2005. (Cited page 49.)

[82] John Franco and Alain Van Gelder. A perspective on certain polynomial-time solvable classes of satisfiability. *Journal of Discrete Applied Mathematics*, 125:177–214, 2003. (Cited pages 49, 50, 55, 56, 57, 59, 84 and 121.)

[83] John S. Schlipf, Fred S. Annexstein, John V. Franco, and Ramjee P. Swaminathan. On finding solutions for extended Horn formulas . (Cited pages 49, 50 and 121.)

[84] Ondrej Cepek, Petr Kucera, and Václav Vlcek. Properties of SLUR formulae. In *Proceedings of the 38th Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2012)*, volume 7147 of *Lecture Notes in Computer Science*, pages 177–189. Springer, 2012. (Cited pages 50, 66, 78, 81, 82 and 121.)

[85] J.J. Hebrard and P. Luquet. The Horn basis of a set of clauses. *Journal of Logic Programming*, 34(1), 1998. (Cited pages 51, 52 and 121.)

[86] P. Luquet. *Horn renommage partiel et littéraux purs dans les formules*. PhD thesis, Université d'Caen, 2000. (Cited pages 51, 52 and 121.)

[87] Emmanuel Benoist and Jean-Jacques Hebrard. Ordered formulas. *Les cahiers du GREYC, CNRS - UPRES-A 6072 (search report), ,University of Caen-Basse-Normandie*, (14), 1999. (Cited pages 52, 53, 54, 55 and 121.)

[88] Endre Boros, Peter L. Hammer, and Xiaorong Sun. Recognition of q-Horn formulae in linear time. *Annals of Mathematics and Artificial Intelligence*, 1:21–32, 1990. (Cited pages 55, 56 and 121.)

[89] Hammer P. L. Boros E., Crama Y. and Saks M. A complexity index for satisfiability problems. *SIAM J. Comput.*, 23(1):45–49, 1994. (Cited pages 55, 56 and 121.)

[90] K. Truemper. *Effective Logic Computation*. Wiley, New York, 1998. (Cited page 56.)

[91] S. Szeider. Generalizations of matched CNF formulas. *Ann. Math. Artif. Intell.*, 43(1):223–238, 2005. (Cited pages 57 and 58.)

[92] Oliver Kullmann. Investigations on autark assignments. *Discrete Applied Mathematics*, 107(1–3):99–137, 2000. (Cited pages 59 and 121.)

[93] Oliver Kullmann. Lean clause-sets: generalizations of minimally unsatisfiable clause-sets. *Discrete Applied Mathematics*, page 130:209–249, 2003. (Cited pages 59 and 121.)

[94] Hans van Maaren. A short note on some tractable cases of the satisfiability problem. *Information and Computation*, 158(2):125–130, 2000. (Cited pages 59, 60 and 84.)

[95] Jose Portillo R. and Jose Rodrigues I. Purl: A polynomial-time algorithm for some polynomial satisfiability classes. *Journal of Algorithms*, 2011. (Cited pages 61 and 121.)

[96] E. Goldberg. Proving unsatisfiability of CNFs locally. *J. Autom Reasoning.*, 28(5):417–434., 2002. (Cited page 61.)

[97] Donald E. Knuth. Nested satisfiability. *Acta Inf.*, 28(1):1–6, 1990. (Cited page 62.)

[98] Pierre Hansen, Gerard plateau, and Brigitte Jaumard. An extension of nested satisfiability. *RUTCOR Research Reports*, pages 29–93, 1993. (Cited pages 62 and 63.)

[99] Amitabha Roy. Fault tolerant boolean satisfiability. *Journal of Artificial Intelligence Research*, 25:503–527, 2006. (Cited page 64.)

[100] Joost P. Warners and Hans van Maaren. Recognition of tractable satisfiability problems through balanced polynomial representations. *Discrete Applied Mathematics*, 99:229–244, 2000. (Cited page 64.)

[101] Stefan Porschen, Ewald Speckenmeyer, and Xishun Zhao. Linear CNF formulas and satisfiability. *Discrete Applied Mathematics*, 157(5):1046–1068, 2009. (Cited page 65.)

[102] Stefan Porschen and Ewald Speckenmeyer. A CNF class generalizing exact linear formulas. In *Volume 4996 of the series Lecture Notes in Computer Science*, pages 231–245, 2008. (Cited page 65.)

[103] Dominik Scheder. Satisfiability of Almost Disjoint CNF Formulas. *CoRR abs/0807*, 2008. (Cited page 65.)

[104] Ingo Schiermeyer. The k-SATISFIABILITY problem remains NP-complete for dense families. *Discrete Mathematics*, 125(1-3):343–346, 1994. (Cited page 65.)

[105] S. Szeider. Minimal unsatisfiable formulas with bounded clause-variable difference are fixed-parameter tractable. *J. Comput. Syst. Sci.*, 69:656–674, 2004. (Cited page 66.)

[106] Václav Vlcek, Tomas Balyo, Stefan Gurský, and Petr Kucera. On hierarchies over the SLUR class. In *International Symposium on Artificial Intelligence and Mathematics (ISAIM 2012) (on-line proceedings http://www.cs.uic.edu/bin/view/Isaim2012/AcceptedPapers)*, 2012. (Cited pages 66, 78, 79, 80, 81, 82, 83 and 121.)

[107] Matthew Gwynne and Oliver Kullmann. Generalising and unifying SLUR and unit-refutation completeness. *SOFSEM*, pages 220–232, 2013. (Cited pages 66, 78, 80, 81, 83 and 121.)

[108] Matthew Gwynne and Oliver Kullmann. Generalising unit-refutation completeness and SLUR via nested input resolution. *J. Autom. Reasoning*, 52(1):31–65, 2014. (Cited pages 66, 78, 80, 81, 83 and 121.)

[109] Stefan Andrei, Gheorghe Grigoras, Martin C. Rinard, and Roland H. C. Yap. A hierarchy of tractable subclasses for SAT and counting SAT problems. *SYNASC*, pages 61–68, 2009. (Cited pages 66 and 83.)

[110] S. Yamasaki and S. Doshita. The satisfiability problem for a class consisting of Horn sentences and some non-Horn sentences in propositional logic. *Information and Control*, 59(1-3):1–12, 1983. (Cited page 67.)

[111] V. Arvind and S. Biswas. An o(n2) algorithm for the satisfiability problem of a subset of propositional sentences in CNF that includes all Horn sentences. *Information Processing Letters*, 24(1):67 – 69, 1987. (Cited page 67.)

[112] H.K.Buning. On generalized Horn formulas and k-resolution. *Theoretical Computer Science*, 116(2):405–413, 1993. (Cited pages 67 and 69.)

[113] A. del Val. Tractable databases: How to make propositional unit resolution complete through compilation. In *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning (KR'94)*, pages 551–561, 1994. (Cited page 81.)

[114] Balasim Al-Saedi, Eric Grégoire, Bertrand Mazure, and Lakhdar Sais. Extensions and Variants of Dalal's Quad Polynomial Fragments of SAT. In *proceedings of the 26th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2014)*, pages 446–452, Limassol, Cyprus,November 10-12, 2014. (Cited page 89.)

[115] Mohammad Al-Saedi, Éric Grégoire, Bertrand Mazure, and Lakhdar Saïs. About some UP-based polynomial fragments of SAT. In *ISAIM*, 2014. (Cited page 95.)

[116] O. Dubois, P. André, Y. Boufkhad, and Y. Carlier. *Second DIMACS implementation challenge: cliques, coloring and satisfiability*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, chapter SAT vs. UNSAT, pages 415–436. American Mathematical Society, 1996. (Cited page 95.)

[117] Chu Min Li and Anbulagan. Look-ahead versus look-back for satisfiability problems. In *Proceedings of the Third International Conference on Principles and Practice of Constraint Programming (CP'97)*, volume 1330 of *Lecture Notes in Computer Science*, pages 341–355. Springer, 1997. (Cited page 95.)

[118] Zvi Galil. On the validity and complexity of bounded resolution. In *STOC*, pages 72–82, 1975. (Cited pages 97, 98 and 121.)

[119] Eli Ben-Sasson and Avi Wigderson. Short proofs are narrow—resolution made simple. *J. ACM*, 48(2):149–169, March 2001. (Cited page 97.)

[120] Paul Beame and Toniann Pitassi. Simplified and improved resolution lower bounds. In *IN PROCEEDINGS OF THE 37TH IEEE FOCS*, pages 274–282. IEEE, 1996. (Cited page 97.)

[121] Zvi Galil. On resolution with clauses of bounded size. *SIAM J. Comput.*, 6(3):444–459, 1977. (Cited page 98.)

[122] Mohammad Al-Saedi, Olivier Fourdrinoy, Éric Grégoire, Bertrand Mazure, and Lakhdar Saïs. About some UP-based polynomial fragments of SAT. In *proceedings of the 27th IEEE International Conference on Tools with Artificial Intelligence (IEEE-ICTAI 2015),Vietri sul Mare, Italy*, pages 405–412, November 9-11, 2015. (Cited page 101.)

[123] Balasim Al-Saedi, Olivier Fourdrinoy, Éric Grégoire, Bertrand Mazure, and Lakhdar Saïs. About some UP-based polynomial fragments of SAT. *Annals of Mathematics and Artificial Intelligence*, pages 1–20, 08 April 2015. (Cited pages 101 and 109.)

[124] O Fourdrinoy, É. Grégoire, B. Mazure, and L. Saïs. Eliminating redundant clauses in SAT instances. In *Proceedings of the The Fourth International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'07)*, volume 4510 of *Lecture Notes in Computer Science*, pages 71–83. Springer, 2007. (Cited pages 101 and 104.)

[125] N. Eén and N. Sörensson. An extensible SAT-solver. In *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*, pages 502–518, 2003. (Cited page 106.)

[126] Mohammad Al-Saedi, Éric Grégoire, Bertrand Mazure, and Lakhdar Saïs. About some UP-based polynomial fragments of SAT. In *proceedings of International Symposium on Artificial Intelligence and Mathematics. Fort Lauderdale, FL.*, January 6-8, 2014. (Cited page 109.)

[127] Shlomo Hoory and Stefan Szeider. Computing unsatisfiable k-SAT instances with few occurrences per variable. *Theoretical Computer Science*, 337(1-3):347–359, 9 June 2005. (Cited page 109.)

[128] J. Kratochvil, P. Savicky', and Z. Tuza. One more occurrence of variables make satisfiability jump from trivial to NP-complete. *Acta Informatica*, 30:397–403, 1993. (Cited page 109.)

[129] P. Savicky and J. Sgall. DNF tautologies with a limited number of occurrences of every variable. *Theoretical Computer Science*, 238:495–498, 2000. (Cited page 109.)

[130] O. Dubois. On the r,s-SAT satisfiability problem and a conjecture of Tovey. *Discr. Appl. Math.*, 26:51–60, 1990. (Cited pages 109, 110 and 111.)

[131] J. Stribrna. Between combinatorics and formal logic. Thesis (m.s.), Charles University, Prague, 1994. (Cited page 109.)

[132] P. Berman, M. Karpinski, and A.D. Scott. Approximation hardness and satisfiability of bounded occurrence instances of SAT. Technical report, TR03-022,Electronic Colloquium on Computational Complexity (ECCC), 2003. (Cited page 109.)

[133] Belaid Benhamou. Satisfiability and matchings in bipartite graphs: relationship and tractability. *RACSAM*, 98:55–63, 2004. (Cited page 111.)

[134] Marek Karpinski Piotr Berman and Alexander D. Scott. Computational complexity of some restricted instances of 3-SAT. (Cited pages 111 and 112.)

[135] K. Iwama and K. Takaki. Satisfiability of 3CNF formulas with small clause/variable ratio. In *Satisfiability Problem: Theory and Applications, Dingzhu Du, Jun Gu, and Panos M. Pardalos, eds, American Mathematical Society, DIMACS Series in Discrete Mathematics and Theoretical Computer Science(35)*, page 315–333, 1997. (Cited page 111.)

Résumé

La représentation des connaissances et les problèmes d'inférence associés restent à l'heure actuelle une problématique riche et centrale en informatique et plus précisément en intelligence artificielle. Dans ce cadre, la logique propositionnelle permet d'allier puissance d'expression et efficacité. Il reste que, tant que P est différent de NP, la déduction en logique propositionnelle ne peut admettre de solutions à la fois générales et efficaces. Dans cette thèse, nous adressons le problème de satisfiabilité et proposons de nouvelles classes d'instances pouvant être résolues de manière polynomiale.

La découverte de nouvelles classes polynomiales pour SAT est à la fois importante d'un point de vue théorique et pratique. En effet, on peut espérer les exploiter efficacement au sein de solveurs SAT. Dans cette thèse, nous proposons d'étendre deux fragments polynomiaux de SAT à l'aide de la propagation unitaire tout en s'assurant que ces fragments demeurent reconnus et résolus de manière polynomiale.

Le premier résultat de cette thèse concerne la classe Quad. Nous avons établi certaines propriétés de cette classe d'instances et avons étendu cette dernière de manière à s'abstraire de l'ordre imposé sur les littéraux. Le fragment obtenu en remplaçant cet ordre par différents ordres sur les clauses, conserve la même complexité dans le pire cas. Nous avons également étudié l'impact de la résolution bornée et de la redondance par propagation unitaire sur cette classe.

La seconde contribution concerne la classe polynomiale proposée par Tovey. La propagation unitaire est une nouvelle fois utilisée pour étendre cette classe. Nous comparons le nouveau fragment polynomial obtenu à deux autres classes basées également sur la propagation unitaire : Quad et UP-Horn. Nous apportons également une réponse à une question ouverte au sujet des connexions de ces classes. Nous montrons que UP-Horn et d'autres classes basées sur la propagation unitaire sont strictement incluses dans $\bigcup$Quad qui représente l'union de toutes les classes Quad obtenues par l'exploitation de tous les ordres sur les clauses possibles.

Mots-clés : Logique propositionnelle, SAT, classes polynomiales.

Abstract

Knowledge representation and reasoning is a key issue in computer science and more particularly in artificial intelligence. In this respect, propositional logic is a representation formalism that is a good trade-off between the opposite computational efficiency and expressiveness criteria. However, unless $P = NP$, deduction in propositional logic is not polynomial in the worst case. So, in this thesis we propose new extensions of tractable classes of the propositional satisfiability problem. Tractable fragments of SAT play a role in the implementation of the most efficient current SAT solvers, many of these tractable classes use the linear time unit propagation (UP) inference rule. We attempt to extend two of currently-known polynomial fragments of SAT thanks to UP in such a way that the fragments can still be recognized and solved in polynomial time. A first result focuses on Quad fragments: we establish some properties of Quad fragments and extend these fragments and exhibit promising variants. The extension is obtained by allowing Quad fixed total orderings of clauses to be accompanied with specific additional separate orderings of maximal sub-clauses. The resulting fragments extend Quad without degrading its worst-case complexity. Also, we investigate how bounded resolution and redundancy through unit propagation can play a role in this respect. The second contribution on tractable subclasses of SAT concerns extensions of one well-known Tovey's polynomial fragment so that they also include instances that can be simplified using UP. Then, we compare two existing polynomial fragments based on UP: namely, Quad and UP-Horn. We also answer an open question about the connections between these two classes: we show that UP-Horn and some other UP-based variants are strict subclasses of $\bigcup$Quad, where $\bigcup$Quad is the union of all Quad classes obtained by investigating all possible orderings of clauses.

Keywords: Propositional logic, SAT, Tractable classes.