Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Departamento de Ingeniería Eléctrica

Sección de Computación

Satisfactibilidad de problemas no clausales con estructura Horn

Tesis que presenta

Edgar Altamirano Carmona

para obtener el Grado de

Doctor en Ciencias

en la Especialidad de

Ingeniería Eléctrica

Codirectores de la Tesis:

Ana María Martínez Enríquez

Gonzalo Escalada Imaz

México, D.F. **Agosto del 2005**

CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS
DEL INSTITUTO POLITÉCNICO NACIONAL

Department of Electrical Engineering

Computer Science Section

On Non-clausal Horn-like Satisfiability Problems

Thesis presented by:

Edgar Altamirano Carmona

to obtain the grade of

Doctor es Science

Electrical Engineering

Speciality

Co-directors:

Ana María Martínez Enríquez

Gonzalo Escalada Imaz

México, D.F. **August 9th, 2005**

Contents

Resumen

Muchas aplicaciones en informática requieren representar conocimiento y razonar con fórmulas en formas no clausales.

Esta tesis está dedicada a probar que existen varias clases de fórmulas en formas no clausales que son estrictamente lineales en el caso de la lógica proposicional y casi-lineales en el caso de la lógica multivaluada.

Nuestra contribución científica puede ser considerada desde dos puntos de vista: teórico y práctico.

En cuanto al aspecto teórico, nuestros resultados amplían el campo de la tratabilidad no clausal. Así, en un primer lugar hemos definido varias clases de fórmulas no-clausales en forma normal negada con una estructura de Horn. En segundo lugar, hemos establecido un cálculo lógico para cada una de estas clases, consistente en un conjunto de reglas de inferencia que probamos que forman un cálculo sólido y refutacionalmente completo. En tercer lugar, hemos diseñado algoritmos estrictamente lineales para el caso de la lógica proposicional y hemos desarrollado algoritmos casi-lineales para el caso de la lógica multivaluada. Estos algoritmos resuelven eficientemente el problema de la satisfactibilidad en cada clase correspondiente.

En cuanto al aspecto práctico, como las fórmulas mantienen una estructura de Horn, son de relevante interés en varias aplicaciones como por ejemplo las provenientes de los sistemas basados en reglas. De hecho, las reglas y preguntas en muchas aplicaciones reales requieren representar y razonar con un lenguage más rico que el ofrecido por el lenguaje de fórmulas de Horn. En este sentido, nuestras fórmulas absorben las fórmulas de Horn como un caso particular. Además, nuestras fórmulas son lógicamente equivalentes a fórmulas de Horn pero utilizan un número exponencialmente inferior de símbolos. Por lo tanto, como los algoritmos descritos corren en tiempo lineal o casi lineal sobre estas clases de problemas, la ganancia experimentada puede ser de un orden exponencial con respecto a los algoritmos conocidos ejecutándose sobre las clásicas fórmulas de Horn.

Abstract

Many applications in Computer Science require to represent knowledge and reason with non-clausal form formulas. However, most of the advances in tractable reasoning are related only to clausal (CNF) formulas.

This thesis is devoted to prove that several classes of non-clausal formulas are strictly linear in the two-valued paradigm of the logic and almost linear in the case of many-valued regular non-clausal Horn-like formulas.

Our scientific contribution can be viewed from two points of view: theoretical and practical.

On the theoretical side, our results aims at pushing further the frontiers of non clausal tractability. Thus, we firstly have defined several classes of non-clausal formulas in Negation Normal Form having a Horn-like shape. Secondly, we have established a Logical Calculus for each one of these classes, consisting of sets of inference rules which we prove they are sound and refutationally complete. In third place, we have designed several strictly linear algorithms for the cases of bi-valued paradigms and we also have developed several almost linear algorithms for the many-valued regular cases. These algorithms resolve efficiently the satisfiability problem in their related classes of formulas.

On the practical side, as the non-clausal formulas keep a Horn-like structure, they are of relevant interest in many and very heterogenous applications as for instance all those based on Rule Based Systems. Indeed, rules and questions of many real applications require to represent and to reason with a richer language than the Horn formulas language. In this sense, our formulas absorb the Horn language as a particular case. Additionally, our formulas represent logically equivalent classical Horn problems but with exponentially less symbols. Hence, as the described algorithms run in linear or an almost linear time on these classes, the gain of execution time could be of an exponential order with respect to the known algorithms running over classical Horn formulas.

Acknowledgments

I like to thank to my research advisors, Drs. Ana María Martínez Enríquez and Gonzalo Escalada-Imaz, for the interest and dedication they took in my work. Their patient considerations guiding me through every step of my research have been of great value throughout the development of this thesis. I am deeply in debt to them for suggesting the topic to me, for providing, supporting and for the arguments which were definitive ones in the successful development of this work. Their support helped to me in improving my technical readings, research writings and presentation skills.

I want to give my all my gratitude to the Computation Section of the Electrical Engineering Department of the Advanced Study and Research Center of the National Polytechnical Institute in Mexico city (CINVESTAV-IPN), which gave me the opportunity to study and develop my Doctoral research in the field of Artificial Intelligence. I like to give special agreements to the researchers, administrative staff, workers and colleagues of the Computation Section, whose warm and friendship were invaluable for me.

Also, I like to give very special thanks to the Artificial Intelligence Research Institute of the Spanish Scientific Research Council (IIIA-CSIC) at Barcelona, Spain, for giving me the opportunity of doing my own research in this charming place. I must tell that this dissertation was benefitted from comments and suggestions made by people in the Institute. In particular, the research seminars I was allowed to give with the specific purpose of explain the development of my work, were very rich in useful feedbacks.

Chapter 1

Introduction

1.1 Motivation

In some practical applications of Computer Science, the well-known Normal Forms CNF (conjunctive normal form) and DNF (disjunctive normal form) do not provide a natural framework to represent knowledge and to reason. In fact, performing inferences efficiently with formulas whose forms are non restricted to the classical ones is a matter of major interest in many practical applications inside very heterogeneous areas such as Expert Systems, Deductive Data Bases, Hardware Design, Automated Software Verification, Symbolic Optimization, Logic Programming, Automated Theorem Proving, Petri Nets, Truth Maintenance Systems, etc.

However, most of the existing efficient proof methods are designed to work with CNF formulas. So, it is a common practice to translate knowledge representations from general forms to CNF's [26, 80]. This transformation was originally proposed in 1970 by Tseitin [84] who published the first algorithm. Now, it is known that any propositional formula can be translated to another equivalent formula in the CNF form applying only the Morgan's rules or other optimized algorithms [80]. The resulting formula is no necessarily unique except if the connectives in the original formula are only $\neg$, $\wedge$ and the or-exclusive connective [54]. The translation process occurs in polynomial time if some auxiliary propositions are allowed in the CNF formula, but it takes exponential time if they are not allowed [37]. A discussion of the advantages of the CNF formulas in the context of theorem proving can be found in [74, 72] In the case of other logics, Henschen et all [51] included the case for first-order logic, Mints [68] covered the cases for modal and intuitionistic logics and finally, Hähnle [47] investigated the problem of translating arbitrary finitely valued logics to short CNF signed formulas.

Currently, two transformations are applied, one preserve the logical equivalence and the other only the satisfiability equivalence.

1. In the first case, the translation cannot skip the explosion of the number

of symbols due to the $\wedge/\vee$ distribution operation and thus the size of the resulting CNF formula can increase exponentially.

2. The other approach consists in modifying the formula by introducing artificial literals [84] aiming at preserving the satisfiability relation. This second line of solution has two strong drawbacks. First, the logical equivalence relation is lost which could be invalid for certain applications. Second, to solve the SAT problem two procedures are required: the first one transforms the original formula into a CNF formula, and the second one, taking as its input the translated CNF formula which is bigger than the original one, applies properly the satisfiability test.

Hence, processing directly the non-clausal formula in an appropriated way arises as the most efficient approach of solving non-clausal SAT problems.

1.2 Contributions

On the theoretical side, our contribution described here aims at pushing further the frontiers of non-clausal tractability. Thus, we firstly have defined a new class of formulas in Negation Normal Form having a Horn-like shape. In this sense, the proposed formulas absorb the Horn language as a particular case. Secondly, we have established a set of inference rules which are sound and refutationally complete. In third place, we have designed respectively, strictly linear algorithms to solve the propositional satisfiability problem and almost linear algorithms to solve the many-valued satisfiability problem.

On the practical side, as the formulas keep a Horn-like structure, they are of relevant interest in such applications as for instance those based in Rule Based Systems. Indeed, the rules and the questions of many real applications require to represent and reason with a richer language than the Horn formulas language. The proposed formulas represent logically equivalent pure Horn problems but with exponentially less symbols. Hence, as the described algorithm runs in linear time on this class, the gain of time can be of an exponential order with respect to the known linear algorithms running on the Horn formulas.

1.3 Structure of the thesis

The thesis is organized in six chapters, whose contents are summarized below:

Chapter 1. Introduction
In this chapter, we describe first the motivations which lead us to work on this research. Second, we mention briefly the importance we believe our results and scientific contributions have. Finally the structure of this thesis is described.

Chapter 2. Propositional Logic: Basic Concepts
In this chapter, we give the basic propositional concepts employed in the next

chapters of this thesis. This chapter is introduced aiming at a self contained thesis memory.

Chapter 3. The antecedents: The Horn-CNF SAT problem
In this chapter, we set out some basic definitions and the terminology that will be used in the thesis. In the first section we describe syntax, semantic, inference rules and a strictly linear algorithm for solving the classical two-valued Horn-CNF SAT problem. In the second section we do the same but for the many-valued Horn-CNF-SAT problem and finally in the third section we deal with the case of the SAT problem for Regular Horn-CNF propositional formulas.

Chapter 4. The propositional Horn-NNF SAT problem
In this chapter, we prove that the Horn-NNF-SAT problem of the two-valued propositional logic can be solved in strictly linear time. In order to proof this result, we first define syntax, semantic and inference rules for the logical system of the Horn-NNF formulas we introduce here. Next, we develop a strictly linear algorithm to solve the SAT problem in this class of formulas. In order to prove the correctness of the algorithm we apply a methodology which proves progressively the logical properties of the final and complicated linear algorithm. This methodology is firstly applied to a sub-case of the Horn-NNF-SAT problem which we call the Simple-Horn-NNF-SAT problem and afterwards it is extended to the general Horn-NNF-SAT problem.

Chapter 5. The Regular Horn-NNF SAT problem
In this chapter, we define two Many-valued Non-clausal Horn-like SAT problems: the Regular Simple-Horn-NNF SAT problem and the Regular Horn-NNF SAT problem. These problems are solved efficiently in $O(n \cdot log(n))$ and $O(n^2)$ time respectively. Thus, we have generalized some existing results about many-valued clausal tractability to the more general many-valued non-clausal framework. The non-clausal formulas considered here could be of significant interest in applications because they present a Horn-like structure. An important advantage of the proposed method is that it does not need to transform the original formula. Indeed, it processes the original formula preserving in this way all its logical properties contrarily to what happens when the formula is transformed to clausal forms by introducing artificial literals.

Chapter 6. Conclusions
In this chapter, we summarize the previous chapters and conclude on the contributions provided in this thesis.

Chapter 2

Propositional Logic: Basic Concepts

2.1 Introduction

The current chapter defines the basic concepts of propositional Logic employed in this thesis. This material can be found in any classical book of Logic as for instance [58], [20], [66] and [29].

Mathematical Logic is the science addressed to study the valid reasonings. We say that a reasoning is valid if each time that the premisses are true the conclusion is necessarily true. Logic does not study the truth or the falseness of the premisses and the conclusion in an isolated way, rather it studies the relation between the truth or falseness of the premisses and the truth or falseness of the conclusion. In other words, Logic is interested in how the truth is propagated from the premisses towards the conclusion.

The validity of a reasoning has no relation at all with the topic treated by the premisses and the conclusion, nor with its truth or falseness. The validity, from a logic point of view, depends on the structure of the reasoning. One valid reasoning has the following form:

If A then B A therefore B

Thus, if we substitute A and B by any sentence, we obtain a valid reasoning. The previous reasoning is not the only valid one. There are other ones, as for instance,

If A then B If B then C therefore if A then C

Logicians build systems that formalize the notion of valid reasoning. The first step, in order to build these systems, is to define a formal language suitable to represent the premisses and the conclusions as sentences of this language. This part is denominated *syntax of the logic.* Yet, as we are interested in how the truth is propagated from the premisses to the conclusion, we have to define when a sentence of the formal language is true or false for a particular interpretation. This part is concerned with the *semantics* or *models theory.* In addition to the syntax and the semantics, a logic is associated with a *proof theory.* Proof Theory studies the procedures that can be mechanized (and their properties) and that permit to find whether a reasoning is valid by means of a symbolic manipulation of the sentences of the language, without taking into account their semantics.

The logic that we will study in this chapter is the Propositional Logic known as well as Calculus of Predicates of order zero (CP_0).

2.2 The CP_0 language

The CP_0 language will be used to represent sentences of the natural language of which we can establish they are true of they are false. This kind of sentences are called *declarative sentences* and are used to make reasonings. The simple declarative sentences are represented by the capital letters P, Q and R and they will be denominated *atoms.* The composed declarative sentences will be represented as a combination of atoms and connectives. The connectives are a formalization of the particles *not, and, or, conditional construction if...then...* and the bi-conditional *if and only if*:

- $\neg$ denotes the particle *not* and it is denominated *negation*
- $\wedge$ denotes the particle *and* and it is denominated *conjunction*
- $\vee$ denotes the particle *or* and it is denominated *disjunction*
- $\rightarrow$ denotes the construction *if...then...* and it is denominated *conditional*
- $\leftrightarrow$ denotes the construction *if and only if* and it is denominated *bi-conditional*

The CP_0 language is a formal language and it will be defined by giving the alphabet of its symbols and the grammatical rules needed for the construction of formulas or sentences.

Definition 2.2.1 *The alphabet Σ of the CP_0 language has the following components:*

- *A set of atom symbols* $\{P,Q,R,P_1,P_2,P_3, \dots \}$
- *A set of connective symbols* $\{ \neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$
- *A set of auxiliary symbols* $\{(,)\}$

Σ={ $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$,(,),P,Q,R,P_1,P_2,P_3, ...}

Definition 2.2.2 *(**sentence**)Given an alphabet Σ*

- *Every atom of Σ is a sentence*
- *If A is a sentence then $\neg A$ is also a sentence*
- *If A and B are sentences, then $(A \wedge B)$, $(A \vee B)$, $(A \rightarrow B)$ and $(A \leftrightarrow B)$ are also sentences.*
- *There are not other sentences*

Notation The letters A, B, C, A_1, A_2, A_3, ..., denote sentences and the Greek letters Γ, Δ, Θ, Γ_1, Γ_2, Γ_3, ... denote set of sentences.

The set of all sentences that can be constructed from the alphabet Σ employing the previous rules form the CP_0 language. It can be remarked that we do not define a language but a family of languages since the set of atoms is not prefixed. We will note $\mathcal{L}(\mathcal{P})$ a propositional language that contains $\mathcal{P}$ as the set of atom symbols. Usually, $\mathcal{P}$ is known as the set of *propositional variables.*

For the sake of readability, we can relax the notation with the following conventions:

1. To remove the external parenthesis. For example, $P1 \wedge P2$ is read as $(P1 \wedge P2)$
2. To associate a priority to each connective in decreasing order as follows:

$$\leftrightarrow, \rightarrow, \vee, \wedge, \neg$$

   in a way that the connective with higher priority has more scope. For instance, $\neg P_1 \wedge P_2$ is read as $((\neg P_1) \wedge P_2)$ and $Q \rightarrow R \leftrightarrow P$ is read as $((Q \rightarrow R) \leftrightarrow P)$

3. When there are several occurrences of a same connective we follow the left associativity rule. For example, $P_1 \rightarrow P_2 \rightarrow P_3$ is read as $((P_1 \rightarrow P_2) \rightarrow P_3)$.

2.3 Semantics

Semantics refer to the study of the relationship between a formal language and its interpretations, by using the concept of truth of a sentence as a bridge concept. From a syntax point of view, a language is a set of sentences, where the sentences are formed by a set of connectives and symbols of atoms. In other words, sentences are strings of characters built according to certain grammatical rules and they have no meaning.

To attribute meaning to a symbol of an atom consists of assigning to it one of the two truth values: true or false. The true value is assigned when the simple declarative sentence denoting the atom is believed to be true in our reasoning context; otherwise, the false value is assigned. The meaning of a sentence formed by two more simple sentences and a connective is defined as a function of the truth value of the simple sentences. As the sentences are built by combining atoms and connectives, if we know the truth values of the atoms involved in the sentence then we can know the truth value of the sentence, since the meaning of the connective does not depend on the context reasoning.

Definition 2.3.1 *Let $\mathcal{L}(\mathcal{P})$ be a CP_0 language, where $\mathcal{P}$ is the set of propositional variables. An interpretation of $\mathcal{L}(\mathcal{P})$ is a function $\mathcal{I}$ with domain the set of propositional variables $\mathcal{P}$ and with rank the set of truth values $\{T,F\}$*

Hence, an interpretation consists of assigning a truth value to each propositional variable. If A is a sentence of a language $\mathcal{L}(\mathcal{P})$ for which an interpretation $\mathcal{I}$ has been defined and P_1, P_2, ...,P_n are the atoms involved in A, the function $\mathcal{I}$ restricted to P_1, P_2, ...,P_n is an interpretation of A.

The next definition establishes the meaning of the connectives and gives the rules that permit to know the truth value of a sentence assigned by an interpretation.

Definition 2.3.2 ***Semantics of the sentences*** *Let A and B be sentences. The meaning of a sentence in an interpretation is defined as follows:*

- *If A is an atom then A is true in $\mathcal{I}$ if $\mathcal{I}$ assigns the value T to A ($\mathcal{I}(A)$=T). Otherwise A is false*
- *$\neg A$ is true in $\mathcal{I}$ when A is false, and false when A is true*
- *$(A \land B)$ is true in $\mathcal{I}$ when A and B are true. Otherwise it is false*
- *$(A \lor B)$ is true in $\mathcal{I}$ when A or B are true. Otherwise it is false*
- *$(A \to B)$ is true in $\mathcal{I}$ when A is false or B is true. Otherwise it is false*
- *$(A \leftrightarrow B)$ is true in $\mathcal{I}$ when A and B have the same truth values. Otherwise it is false.*

The previous rules can be expressed by means of the so called truth tables.

A	B	$A \land B$	$A \lor B$	$A \to B$	$A \leftrightarrow B$
V	V	V	V	V	V
V	F	F	V	F	F
F	V	F	V	V	F
F	F	F	F	V	V

When the connective is the negation then there is only one sentence involved.

A	$\neg A$
V	F
F	V

Note that, if a sentence has n different atoms, then there are 2^n possible interpretations.

Definition 2.3.3 ***Model of a sentence.*** *Let $\mathcal{I}$ be an interpretation and A a sentence. $\mathcal{I}$ is a model of A or $\mathcal{I}$ satisfies A, noted $\models_{\mathcal{I}} A$, iff $\mathcal{I}$ assigns the value true to A. Otherwise, $\mathcal{I}$ falsifies A and it will be noted $\not\models_{\mathcal{I}} A$.*

An interpretation $\mathcal{I}$ is a model of a set of sentences Γ iff it is a model of every sentence. Note that if Γ=$\{A_1$,...,$A_n\}$, then $\mathcal{I}$ is a model of Γ iff it is a model of $A_1 \wedge ... \wedge A_n$

2.4 Classes of sentences

In this section we define the concepts of valid sentence and unsatisfiable sentence. These sentences take always the same truth value for all the interpretations.

Definition 2.4.1 ***(Valid sentence or tautology)*** *A sentence A is valid iff all the possible interpretations satisfy A. Also it is called tautology and it is noted $\models A$. In the opposite case A is invalid and it is noted $\not\models A$.*

A possible manner of checking whether a sentence is a tautology consists of building the truth table and checking whether the sentence is evaluated to true in any interpretation.

Example 2.4.1 *Let us check that $\neg A \vee A$ is a tautology ($\models \neg A \vee A$)*

A	$\neg A$	$\neg A \vee A$
V	F	V
F	V	V

Theorem 2.4.1 *If A and $A \rightarrow B$ are tautologies then B is a tautology.*

Definition 2.4.2 ***(Unsatisfiable sentence)*** *A sentence A is unsatisfiable if all the interpretations falsify A. Also A is called inconsistent and it is noted $\models \neg A$. In the opposite case A is satisfiable and it is noted $\not\models \neg A$.*

Example 2.4.2 *Check that $\neg A \wedge A$ is unsatisfiable ($\models \neg(\neg A \wedge A)$)*

A	$\neg A$	$\neg A \wedge A$
V	F	F
F	V	F

Theorem 2.4.2 *A is valid iff $\neg A$ is unsatisfiable.*

This theorem establishes that there are two ways of proving whether a sentence is valid: by either constructing the truth table for A and checking that A is true for all the interpretations or by verifying indirectly whether $\neg A$ is unsatisfiable. The latter is called *refutation*.

A set of sentences $\Gamma = \{A_1, ..., A_n\}$ is satisfiable iff it has at least one model, it is valid iff all the interpretations are models of Γ and it is unsatisfiable iff it has no model. Remark that to prove the satisfiability, validity o unsatisfiability of Γ is equivalent to prove it for the sentence $A_1 \wedge A_2 \ldots \wedge A_n$.

Note that not all the sentences are valid or unsatisfiable sentences. The sentences that are invalid and satisfiable ones are denominated contingent sentences.

Definition 2.4.3 *(**contingent sentence**) A sentence is contingent iff there exist interpretations that satisfy A and interpretations that falsify A.*

The different classes of sentences are modeled by the following figure.

$\models A$ *(valid)*	$\not\models A$ *(invalid)*	$\not\models A$ *(invalid)*
A valid	**A contingent**	**A unsatisfiable**
$\not\models \neg A$ *(satisfiable)*	$\not\models \neg A$ *(satisfiable)*	$\models \neg A$ *(unsatisfiable)*

Fig.1. Types of sentences

2.5 Logical Consequence

In a reasoning process, the premisses and the conclusion are distinguished. A reasoning is valid if each time that the premisses are certain the conclusion is necessarily certain. The concept of logical consequence formalizes the idea of valid reasoning.

Definition 2.5.1 *(**Logical Consequence**) Let $\{A_1, A_2, ..., A_n\}$ be a finite set of sentences and B be a sentence. B is logical consequence of $\{A_1, A_2, ..., A_n\}$ iff for any interpretation $\mathcal{I}$ that satisfies $A_1 \wedge A_2 \wedge ... \wedge A_n$, $\mathcal{I}$ satisfies also B. The logical consequence relationship is noted by $A_1 \wedge A_2 \wedge ... \wedge A_n \models B$.*

If $\Gamma = \{A_1, A_2, ..., A_n\}$ is a finite set of sentences, the relation $A_1 \wedge A_2 \wedge ... \wedge A_n \models B$ is written $\Gamma \models B$.

The next theorems reduce the problem of proving that a sentence is logical consequence of a set of sentences to the problem of proving the validity or unsatisfiability of these sentences.

Theorem 2.5.1 *Let $A_1, A_2, ..., A_n$ and B be sentences. B is logical consequence of $A_1, A_2, ..., A_n$ iff $((A_1 \wedge A_2 \wedge ... \wedge A_n) \rightarrow B)$ is a valid sentence.*

Theorem 2.5.2 *Let $A_1, A_2, ..., A_n$ and B be sentences. B is logical consequence of $A_1, A_2, ..., A_n$ iff $((A_1 \wedge A_2 \wedge ... \wedge A_n) \wedge \neg B)$ is unsatisfiable.*

Example 2.5.1 *A possible manner of verifying $P, P \rightarrow Q \models Q$ is proving that $(P \wedge (P \rightarrow Q)) \rightarrow Q$ is valid. Another different way is verifying that the set of sentences $\{P, P \rightarrow Q, \neg Q\}$ is unsatisfiable. Both checkings can be done with the truth tables.*

2.6 Logical Equivalence

In this section we will study when two sentences are logically equivalent.

Definition 2.6.1 ***(Logically equivalent sentences)*** *Let us A and B two sentences. A and B are logically equivalent iff they have the same models. It is noted $A \equiv B$. The expression $A \equiv B$ is denominated equivalence.*

Next, we give some examples of logical equivalences:

$A \wedge A$	$\equiv$	A	(Idempotence)
$A \vee A$	$\equiv$	A	(Idempotence)
$A \wedge B$	$\equiv$	$B \wedge A$	(Commutativity)
$A \vee B$	$\equiv$	$B \vee A$	(Commutativity)
$(A \wedge B) \wedge C$	$\equiv$	$A \wedge (B \wedge C)$	(Associativity)
$(A \vee B) \vee C$	$\equiv$	$A \vee (B \vee C)$	(Associativity)
$(A \wedge (A \vee B))$	$\equiv$	A	(Absorption)
$(A \vee (A \wedge B))$	$\equiv$	A	(Absorption)
$(A \wedge (B \vee C))$	$\equiv$	$((A \wedge B) \vee (A \wedge C))$	(Distributivity)
$(A \vee (B \wedge C))$	$\equiv$	$((A \vee B) \wedge (A \vee C))$	(Distributivity)
$\neg(\neg A)$	$\equiv$	A	(Double Negation)
$\neg(A \wedge B)$	$\equiv$	$\neg A \vee \neg B$	(De Morgan's Law)
$\neg(A \vee B)$	$\equiv$	$\neg A \wedge \neg B$	(De Morgan's Law)
$(A \wedge B)$	$\equiv$	B, If A is a tautology	(Tautology's law)
$(A \vee B)$	$\equiv$	A, If A is a tautology	(Tautology's law)
$(A \wedge B)$	$\equiv$	A, If A is unsatisfiable	(Unsatisfiability's law)
$(A \vee B)$	$\equiv$	B, If A is unsatisfiable	(Unsatisfiability's law)

Example 2.6.1 *If we need to prove that $P \wedge Q \equiv \neg(\neg P \vee \neg Q)$, we begin with $P \wedge Q$ and apply the double negation law on the complete formula. Then, we obtain $P \wedge Q \equiv \neg\neg(P \wedge Q)$. Afterwards, we apply the De Morgan's law to the sub-sentence $\neg(P \wedge Q)$ and we obtain $\neg\neg(P \wedge Q) \equiv \neg(\neg P \vee \neg Q)$.*

2.7 Conjunctive Normal Form

Given a sentence of the CP_0, we want to find a sentence logically equivalent having a syntax structure more regular in such a way that it will be easier to be able to accomplish automatic proofs.

Definition 2.7.1 ***(Literal)*** *A literal is an atom or an atom with a negation symbol.*

Definition 2.7.2 ***(Conjunctive Normal Form)****A sentence A is in Conjunctive Normal Form (CNF) iff it has the form $A_1 \wedge A_2 \wedge ... \wedge A_n$, $n \geq 1$, and each A_i is a disjunction of literals.*

Any sentence can be converted into a CNF employing the following algorithm:

Step 1. To remove the connectives $\rightarrow$ and $\leftrightarrow$ by using the logical equivalences:

$$\begin{array}{ccccc} \mathrm{A} \leftrightarrow \mathrm{B} & \equiv & (A \rightarrow B) \wedge (B \rightarrow A) & (1) \\ A \rightarrow B & \equiv & \neg A \vee B & (2) \end{array}$$

Step 2. To reduce the scope of the negation symbols by employing the following equivalences as many times as might be necessary:

$$\begin{array}{cccc} \neg(\neg A) & \equiv & \mathrm{A} & (3) \\ \neg(A \vee B) & \equiv & \neg A \wedge \neg B & (4) \\ \neg(A \wedge B) & \equiv & \neg A \vee \neg B & (5) \end{array}$$

Step 3. To obtain the conjunctive normal form employing the following equivalences as many times as it will be necessary.

$$\begin{array}{cccc} A \vee (B \wedge C) & \equiv & (A \vee B) \wedge (A \vee C) & (6) \\ A \vee B & \equiv & B \vee A & (7) \\ A \wedge B & \equiv & B \wedge A & (8) \\ (A \vee B) \vee C & \equiv & A \vee (B \vee C) & (9) \\ (A \wedge B) \wedge C & \equiv & A \wedge (B \wedge C) & (10) \end{array}$$

Example 2.7.1 *The CNF of the sentence*

$$(\neg P \vee Q) \rightarrow R$$

is

$$\begin{array}{cl} (\neg P \vee Q) \rightarrow R & \\ \neg(\neg P \vee Q) \vee R & \textit{applying (2)} \\ (\neg\neg P \wedge \neg Q) \vee R & \textit{applying (4)} \\ (P \wedge \neg Q) \vee R & \textit{applying (3)} \\ R \vee (P \wedge \neg Q) & \textit{applying (7)} \\ (R \vee P) \wedge (R \vee \neg Q) & \textit{applying (6)} \end{array}$$

Note that the transformation algorithm preserves the logical equivalence. Therefore, another way of verifying whether two sentences are equivalent consists in checking whether they have the same normal form.

2.8 Proof procedures

We have seen in previous sections that the truth tables let solve the validity problem and the satisfiability problem. However, constructing truth tables is computationally expensive, since if the number of different atoms of a sentence is n, then 2^n rows are required. In addition, the truth tables do not form an intuitive method, because they do not reflect the way of reasoning of humans. In this section, we will see other decision procedures, called *proof procedures*, that let solve the validity problem and the satisfiability problem using the *proof* concept instead of that of interpretation. Until now, to check whether $\Gamma \models A$, we verified whether all the interpretations that satisfy Γ satisfy also A. From now on, to check whether $\Gamma \models A$ we will verify whether there exists a proof of A from Γ.

In order to understand the proof concept, we first need to define the *calculus* concept. A calculus for a language is composed by a set of inference rules. The inference rules formalize schemes of valid reasoning and enable to derive sentences from other sentences of the language. An example of inference rule is the *modus ponens*:

$$\frac{A, A \rightarrow B}{B}$$

This inference rule establishes that given a sentence A and a sentence $A \rightarrow B$ the sentence B can be derived. For instance, if we have the sentence $P \vee Q$ and the sentence $P \vee Q \rightarrow R$ we can derive R. As the inference rules formalize schemes of valid reasoning, we have $P \vee Q, P \vee Q \rightarrow R \models R$.

Notice we have only employed the rule and we have not constructed any truth table.

Another example of inference rule is the *disjunction introduction* rule:

$$\frac{A}{A \vee B}$$

This inference rule establishes that given a sentence A, we can derive the sentence $A \vee B$, where B is any sentence . For instance, if we have the sentence P we can derive the sentence $P \vee Q$. Since rules model valid reasonings, we have $P \models P \vee Q$. Remark we have employed one rule and we have not used any truth table.

If beginning from a set of sentences Γ and applying inference rules, we can derive A, we have found a *proof* of A from Γ and that will be noted $\Gamma \vdash A$. We will call the sequence of sentences generated until finding A a proof of $\Gamma \vdash A$.

We know that checking $\Gamma \models A$ is equivalent to checking that $\Gamma \cup \{\neg A\}$ is unsatisfiable. If beginning from a set of sentences and applying inference rules, we derive the empty set, then we have found a refutation of $\Gamma \cup \{\neg A\}$ and that will be noted $\Gamma \vdash A$.

Notice that the *satisfiability relationship* has been defined as a function of the interpretation concept. Differently, the *deduction relationship* has been defined as a function of the derivation of sentences applying inference rules. Thus, a calculus will be useful if it has the following properties:

1. **Soundness**. Each sentence A deducible from a set of sentences Γ is logical consequence of Γ. Namely, $\Gamma \vdash A \Rightarrow \Gamma \models A$. This property is required to all the calculus, since otherwise we can deduce sentences that are not logical consequence.
2. **Completeness**. For each sentence A that is a logical consequence of a set of sentences Γ there exists a proof of A from Γ. Namely, $\Gamma \models A \Rightarrow \Gamma \vdash A$

In the next section, we study resolution which is a proof procedure by refutation.

2.9 Resolution

In this section, we will study the Resolution calculus. This calculus is characterized by having a unique inference denominated Resolution Principle. To be able to apply resolution, we need to transform the sentences to the clausal form.

2.9.1 Clausal Form

The clausal form is a compact representation of the conjunctive normal form (CNF). We remind that any sentence of the CP_0 can be transformed to a logical equivalent sentence in CNF. Now, we give some few definitions in order to study the transformation from the CNF to the clausal form.

Definition 2.9.1 ***(Dual literal)*** *If L is a literal identic to an atom P then the dual of L, noted L^d, is equal to $\neg P$. Otherwise, if L is of the form $\neg P$, then $L^d = P$.*

Definition 2.9.2 ***(Clause)****. A clause is a finite set of literals that represents the disjunction of these literals.*

Example 2.9.1 *The disjunction $P \vee Q \vee \neg R$ can be represented by the clause $\{P, Q, \neg R\}$.*

A clause is unitary if it contains a unique literal. The clause without literals is the empty clause and it is noted by $\square$. As the empty clause has no literals to be satisfied it is always unsatisfiable and denotes the truth value F.

Definition 2.9.3 ***(Clausal Form)****. Let $A = (L_{1,1} \vee \ldots \vee L_{1,n_1}) \wedge \ldots \wedge (L_{k,1} \vee \ldots \vee L_{k,n_k})$ be a sentence in CNF where each $L_{i,j}$ is a literal. The clausal representation is given by the following set of clauses:*

$$A = \{\{L_{1,1}, \ldots, L_{1,n_1}\}, \ldots, \{L_{k,1}, \ldots, L_{k,n_k}\}\}$$

A clause represents a disjunction. A coma separating two literals in a clause represents a disjunction symbol, meanwhile a coma separating two clauses represents a conjunction symbol.

2.9.2 Resolution Principle

As we have mentioned, the resolution calculus has a unique inference rule, denominated Resolution Principle. The inconvenient of having one unique rule is that sentences must be translated to the clausal form.

Definition 2.9.4 ***(Resolution Principle)****. Let $C_1 = \{L_1, \ldots, L_n\}$ and $C_2 = \{L'_1, \ldots, L'_m\}$ be two clauses. The clause $R = (C1 - \{L_i\}) \cup (C2 - \{L'_j\})$ is a resolvent of C_1 and C_2 if $L_i \in C_1$, $L'_j \in C_2$ and $L_i^d = L'_j$.*

Notice that if $C_1 = \{L\}$ and $C_2 = \{L^d\}$ then the empty clause is obtained.

Example 2.9.2 *Next we give some examples of resolvents:*

$$\begin{array}{ccc} C_1 = \{P, \neg Q, R\} & C_2 = \{Q, R, \neg S\} & R = \{P, R, \neg S\} \\ C_1 = \{Q\} & C_2 = \{\neg P, \neg Q, R\} & R = \{\neg P, R\} \\ C_1 = \{\neg P\} & C_2 = \{P\} & R = \square \end{array}$$

Definition 2.9.5 *Let $\mathcal{C}$ be a set of clauses and C' a clause. A resolution proof of C' from $\mathcal{C}$ is a finite sequence $C_1, C_2, \ldots, C_n$ of clauses such that each C_i, $1 \geq i \geq n$, is a clause of $\mathcal{C}$ or it is a resolvent of two clauses C_j, C_k, $1 \geq j, k < i$, and $C_n = C'$. A proof of $\square$ from $\mathcal{C}$ is denominated a refutation of $\mathcal{C}$.*

Example 2.9.3 *A refutation by resolution of the set of clauses:*

$$\mathcal{C} = \{\{P, Q, \neg R\}, \{\neg P\}, \{P, Q, R\}, \{P, \neg Q\}\}$$

is

$$\begin{array}{ll} C_1 = \{P, Q, \neg R\} & \textit{(clause of } \mathcal{C}\textit{)} \\ C_2 = \{P, Q, R\} & \textit{(clause of } \mathcal{C}\textit{)} \\ C_3 = \{P, Q\} & \textit{(resolvent of } C_1, C_2\textit{)} \\ C_4 = \{P, \neg Q\} & \textit{(clause of } \mathcal{C}\textit{)} \\ C_5 = \{P\} & \textit{(resolvent of } C_3, C_4\textit{)} \\ C_6 = \{\neg P\} & \textit{(clause of } \mathcal{C}\textit{)} \\ C_7 = \square & \textit{(resolvent of } C_5, C_6\textit{)} \end{array}$$

$C_1, C_2, C_3, C_4, C_5, C_6, C_7$ is a refutation of $\mathcal{C}$.

Theorem 2.9.1 ***(Resolution lemma)*** *Let $\mathcal{C}$ be a set of clauses, C_1 and C_2 two clauses of $\mathcal{C}$ and R a resolvent of C_1 and C_2. Then, $\mathcal{C}$ and $\mathcal{C} \cup R$ are logically equivalents.*

Theorem 2.9.2 ***(Resolution theorem)*** *A set of clauses $\mathcal{C}$ is unsatisfiable iff there exists a refutation of $\mathcal{C}$.*

The previous theorem warrants the soundness and completeness of resolution for refutation, namely $\Gamma \models A$ iff $\Gamma, \neg A \vdash \square$. However, it does not warrant $\Gamma \models A$ iff $\Gamma \vdash A$. By the soundness property, one can state that if $\Gamma \vdash A$ then $\Gamma \models A$, but the inverse is not true. That can be seen with a counterexample: $P \models P \vee Q$, however $P \nvdash P \vee Q$. Due to this constraint, it is common to consider that resolution is complete for refutation but it is not complete as a deductive calculus.

Chapter 3

The Antecedents: The Horn-CNF SAT problem

3.1 Introduction

The propositional satisfiability (CNF-SAT) problem is at the core in Computer Science. It was the first NP-complete problem found [24]. Since then, a rather big effort has been done to determine some CNF-SAT islands of tractability with significant repercussions in applications. The most important classes that can be resolved in deterministic polynomial time are: 2-CNF, for which linear algorithms were designed in [15, 36, 30], and Horn-SAT, that admits also linear algorithms as showed in [56, 28, 67, 38, 78, 40]. Several variants of the Horn-SAT problem have been also found out to be solvable in polynomial time: renamable Horn [14, 60], extended Horn [22], CC-balanced [23], SLUR [77] and q-Horn [19, 18].

The Horn-SAT problem is polynomially solvable since the work of Karp [57]. After that, Henschen and Wos [52] showed that if a Horn formula is unsatisfiable, then, there exists a refutation proof employing unit propagation only. Jones and Laaser [56] showed that a direct implementation of this principle leads to an algorithm of quadratic complexity. Later, Dowling and Gallier [28] presented two linear algorithms to resolve the Horn-SAT problem: one with a forward chaining strategy and the second one based on backward chaining. In [78] and [40] it is proved that the backward algorithm is incomplete and not linear respectively. In [67, 38, 32, 3] were proposed different linear versions, all of them based on a forward chaining strategy. A linear and complete algorithm with backward chaining strategy is described in [40].

In this chapter we review three variants of the Horn-CNF SAT problem. First, we review the well known classical (bi-valued) Horn-SAT problem, where syntax, semantic and a complete calculus is stated; also we develop a strictly linear algorithm to solve, using only a stack and counters the related SAT problem. We also prove the complexity and correctness of this algorithm. In the next section we review the case of the Many-valued Horn-SAT problem as it is stated in [33]. Finally, in the last section, the Regular Horn-SAT problem is described.

3.2 Classical Horn-CNF SAT problem

3.2.1 Syntax and semantic

A clause signature is a pair <**P**,**O**> such that **P**$= \{p_1, \ldots, p_n\}$ is a set of n propositional variables and **O**$= \{\wedge, \vee, \neg\}$ is a set of logical connectives.

Definition 3.2.1 *An atomic proposition (or Boolean variable) p is a symbol and* $\mathbf{P}$ *is a finite set of atomic propositions. A literal L is either an atomic proposition $p \in \mathbf{P}$, noted L^+ or its negation $\neg p$, noted L^-. A clause $\mathcal{C}$ is a finite disjunction of literals: $\mathcal{C} = (L_1 \vee \ldots \vee L_m)$. A Horn clause is a clause with at most one non-negative literal. We have three types of Horn clauses: pure negative like $\mathcal{C} = (L_1^- \vee L_2^- \ldots \vee L_m^-)$, pure non-negative as in the unit clause $\mathcal{C} = (L^+)$ and pure Horn clauses as in $\mathcal{C} = (L_1^- \vee L_2^- \ldots \vee L_{m-1}^- \vee L_m^+)$. A propositional CNF formula Γ is a finite conjunction of clauses: $\Gamma = \mathcal{C}_1 \wedge \ldots \wedge \mathcal{C}_n$. A Horn formula is a finite conjunction of Horn clauses.*

Definition 3.2.2 *An interpretation I assigns to each formula Γ one value in the set $\{0, 1\}$ and it satisfies:*

- *A literal p ($\neg p$) iff $I(p) = 1$ ($I(p) = 0$).*
- *A clause $\mathcal{C} = L_1 \vee \ldots \vee L_k$, iff $I(L_i) = 1$, for at least one L_i.*
- *A formula Γ if I satisfies all clauses of the formula.*

An interpretation I is a model *of a formula Γ if satisfies the formula. We say that Γ is* satisfiable *if it has at least one model, otherwise, it is* unsatisfiable.

3.2.2 Logical Calculus

Methods based on unit resolution (UR) [39] are known as the most efficient to resolve the general problem of deduction in Horn logics [32, 35]. In this class of formulas, UR is a sound and complete [52, 53] inference rule.

Definition 3.2.3 *Let $\mathcal{C} = (\neg p \vee \neg p_1 \vee \ldots \vee \neg p_n \vee p_{n+1})$ and $\mathcal{C}' = (p)$ two clauses. Then, the positive unit resolution (PUR) generates the resolvent clause $\mathcal{C}'' = (\neg p_1 \vee \ldots \vee \neg p_n \vee p_{n+1})$.*

Definition 3.2.4 *A deduction of a clause $\mathcal{C}_n$ from an original formula Γ is a succession of clauses $< \mathcal{C}_1, \ldots, \mathcal{C}_n >$ such that for each $1 \leq i \leq n, \mathcal{C}_i \in \Gamma$ or $\mathcal{C}_i$ is a resolvent from two preceding clauses in the succession.*

Unit resolution rule introduces the restriction that one of the clauses must contain only a proposition (positive literal). We will denote the deductive system by $\vdash_{PUR}$. The following results for correctness of the $\vdash_{PUR}$ are well known [52, 53] and easy to prove.

Theorem 3.2.1 Soundness *Let* Γ *be a theory and* $\mathcal{C}$ *a clause, then* $\Gamma \vdash_{PUR} \mathcal{C} \Rightarrow \Gamma \models \mathcal{C}$.

Theorem 3.2.2 Completeness *Let* Γ *be a HORN-formula. If* Γ *is unsatisfiable, then* $\Gamma \vdash_{PUR} \Box$.

The next theorem extends $\vdash_{PUR}$ completeness to atomic clauses.

Theorem 3.2.3 Completeness *Let* Γ *be a HORN-formula and* $\mathcal{C} = (L)$ *a unit clause, then* $\Gamma \models \mathcal{C} \Rightarrow \Gamma \vdash_{PUR} \mathcal{C}$.

A set of clauses is satisfiable if it has at least one model. Concerning this point, there exist two known results:

Proposition 3.2.1 *Let* Γ *a HORN-formula:*

1. $\Gamma^{+} = \emptyset \Rightarrow \exists I,\ I \models \Gamma$
2. $\Gamma^{-} = \emptyset \Rightarrow \exists I,\ I \models \Gamma$

Proof (1) Γ does not contain positive clauses, then all clauses contain at least one negated variable; if we assign to all of them the truth value 0 (false), we will obtain a model. (2) Proof is dual with respect to (1). ■

3.2.3 PPUR Algorithm

The principle of the PPUR algorithm (Propagation of the PPUR inference rule) consists in applying repeatedly the positive unit resolution until either deriving the empty clause (Γ is unsatisfiable) or until there could not be derived new unit clauses (Γ is satisfiable).

The most important characteristics of the PPUR algorithm are the following: its complexity is strictly linear and its presentation given in pseudo code is more simple than the existing and published algorithms like [28, 35, 38, 67].

The associated Data structures are:

Concerning each $\mathcal{C}$ clause
$Cont(\mathcal{C})$: It is a counter which in each moment of the process indicates the number of negative literals $\neg p$ such that the associated propositions p, has not been deduced until that moment.
Initially, $Cont(\mathcal{C})$ indicates the number of negative literals in $\mathcal{C}$.

$Lit.Pos(\mathcal{C})$: Positive literal of the clause $\mathcal{C}$.

Concerning the formula Γ
$Prop(\Gamma)$: Set of propositions p in Γ.

Concerning each proposition p
$Neg(p)$: Set of pointers to clauses containing $\neg p$
$Val(p)$: $Val(p) \in \{0,1\}$ and $Val(p) = 1$ iff p is logical consequence of the formula Γ.

Auxiliary structure
Pila : Data structure of type Stack.

Input: Γ: A Horn formula
Output : *'Satisfiable'* iff Γ is satisfiable.

Algorithm 3.2.1 PPUR(Γ)

```
1.  $Stack \leftarrow \emptyset$;
2.  for $\forall p \in Prop(\Gamma)$ do:
3.      $Val(p) \leftarrow 0$, $Neg(p) \leftarrow \{\}$;
4.  for $\forall \mathcal{C} \in \Gamma$ do:
5.      if $\mathcal{C} \neq \{p\}$ then do:
6.          $Cont(\mathcal{C}) \leftarrow 0$;
7.          if $\exists p \in \mathcal{C}$ then $Lit.Pos(\mathcal{C}) \leftarrow p$;
8.          else $Lit.Pos(\mathcal{C}) \leftarrow Nil$;
9.          for $\forall \neg p_i \in \mathcal{C}$ do:
10.             Increment $Cont(\mathcal{C})$;
11.             Add $\mathcal{C}$ to $Neg(p_i)$;
12.     else: If $Val(p)=0$ then:
13.         $push(p, Stack)$, $Val(p) \leftarrow 1$;
14. while $Stack \neq \emptyset$ do:
15.         $p \leftarrow pop(Stack)$;
16.         for $\forall \mathcal{C} \in Neg(p)$ do:
17.             $Cont(\mathcal{C}) \leftarrow Cont(\mathcal{C}) - 1$;
18.             if $Cont(\mathcal{C}) = 0$ then do:
19.                 if $Lit.Pos(\mathcal{C}) \neq Nil$ then do:
20.                     if $Val(Lit.Pos(\mathcal{C})) = 0$
21.                     then: $push(Lit.Pos(\mathcal{C}), Stack)$;
22.                           $Val(Lit.Pos(\mathcal{C})) \leftarrow 1$;
23.                 else return 'Unsatisfiable';
24. return 'Satisfiable';
```

Theorem 3.2.4 Complexity: *PPUR(Γ) is in O(n).*

Proof. Initialization: (lines 1–12) The first two "for" iterations require a proportional time to the number of propositions and clauses respectively. Third

iteration “for” is proportional to the set of occurrences of negative literals in the non-unitary clauses.
While Cycle: (lines 13-22) The maximum number of iterations of the While instruction is bounded by the number of different propositions in the formula because a same proposition can not be pushed in the Stack more than once. This can be checked by the following facts: 1) In the initialization block a proposition p is introduced in the stack only if exists a unit clause $\mathcal{C} = (p)$. Once it is introduced, it is marked with: $Val(p) = 1$. 2) Next, in the While iteration, before is introduced a proposition in the Stack, it is verified that is not marked to 1, i.e. $Val(p) = 0$. Only in this case is inserted in the Stack and immediately is marked $Val(p) \leftarrow 1$, avoiding that can be pushed for a second time.
Like “Neg(p)” list has an element for each occurrence of $\neg p$, the number of iterations of the ”for” instruction within the While instruction is limited to the number of occurrences $\neg p$ of propositions of the original formula. Consequently, the complexity of the algorithm is linear. ■

Theorem 3.2.5 Soundness and Completeness *The PPUR(Γ) algorithm returns 'Unsatisfiable' iff Γ is unsatisfiable.*

3.3 Many-valued Horn-CNF-SAT problem

We consider as [33] that a many-valued sentence is an ordered pair $(S;\alpha)$ in which S is a classical sentence and α is a truth value attached to the sentence S. The set of truth values is the infinite ordered set formed by the unit rational interval $\Sigma = [0,1]$. This logic can be viewed also as a subclass [48] of the regular logic [46, 50].

3.3.1 Syntax and semantic

Definition 3.3.1 Many-valued formulas *A many-valued literal is a pair of the form $(L;\alpha)$ where L is a classical propositional literal and $\alpha \in \Sigma$. A many-valued clause is a pair of the form $((L_1 \vee \ldots \vee L_m);\alpha)$ where L_i is a classical propositional literal and $\alpha \in \Sigma$. We denote the many-valued empty clause as $\Box$. A many-valued formula is a conjunction of many-valued clauses. Any many-valued formula S containing the many-valued empty clause is denoted by $S_\Box$.*

Example 3.3.1 *An example of a many-valued formula is:* $S = \{(p_1;0.75), (p_3;0.80), ((\neg p_1 \vee \neg p_2 \vee p_4);0.50), ((\neg p_2 \vee \neg p_3 \vee p_1 \vee p_5);0.60), ((\neg p_2 \vee \neg p_5);0.65)\}$.

Definition 3.3.2 Many-valued Horn formula *A many-valued Horn clause is a many-valued clause whose first component has at most one non negated literal. A many-valued Horn formula is a finite conjunction of many-valued Horn clauses.*

Example 3.3.2 *An example of a many-valued Horn formula is:*
$S = \{(p_1;0.75), (p_3;0.80), ((\neg p_1 \vee p_4);0.50), ((\neg p_2 \vee \neg p_3 \vee p_5);0.60), ((\neg p_2 \vee \neg p_5);0.65)\}$.

Definition 3.3.3 Interpretation *An interpretation I is a mapping from the first components of the sentences (Literals, clauses and formulas) to the set Σ of truth values. An interpretation I verifies the following properties:*

$$I(\neg S) = 1 - I(S)$$
$$I(S_1 \vee S_2) = max(I(S_1), I(S_2))$$
$$I(S_1 \wedge S_2) = min(I(S_1), I(S_2))$$

Definition 3.3.4 Satisfiability *An interpretation I satisfies a many-valued clause $C = ((L_1 \vee \ldots \vee L_m); \alpha)$, iff for some literal L_i, $I(L_i) \in [\alpha, 1]$, i.e. iff $I(L_i) \geq \alpha$. A many-valued formula S is satisfiable iff there exists an interpretation that satisfies all the many-valued clauses in S. A many-valued formula that is not satisfiable is unsatisfiable. The empty many-valued clause $\Box$ is always unsatisfiable and the empty many-valued formula $S = \{\}$ is always satisfiable.*

Example 3.3.3 *An interpretation that satisfies the formula in the previous example is for instance $I(p_1) = I(p_3) = I(p_5) = 0.8, I(p_2) = I(p_4) = 0$.*

Proposition 3.3.1 *Let S be any many-valued formula; if S contains two many-valued clauses $(p; \alpha)$ and $(\neg p; \alpha')$ such that $\alpha + \alpha' > 1$ then, S is unsatisfiable*

Proof:
$I(p) \geq \alpha, I(\neg p) \geq \alpha' \Rightarrow I(p) + I(\neg p) \geq \alpha + \alpha'$.
If $\alpha + \alpha' > 1$ then $I(p) + I(\neg p) > 1$.
But $\nexists I$ t.q. $I(p) + I(\neg p) > 1$
because $\forall I, I(p) + I(\neg p) = 1$
according to the first axiom of I.

3.3.2 Logical Calculus

Only one inference rule called many valued positive unit resolution (MPUR) by [33] is needed to obtain a refutation complete calculus for Many valued Horn formulas.

Definition 3.3.5 *The rule MPUR derives from a many valued Horn clause $((\neg p \vee D^- \vee q); \alpha)$ and a unitary many valued Horn clause $(p; \alpha')$ the many valued Horn clause $((D^- \vee q); \alpha)$ provided that $\alpha + \alpha' > 1$. The many valued Horn clause derived from the clauses $(\neg p; \alpha)$ and $(p; \alpha')$ given that $\alpha + \alpha' > 1$ is the empty many valued Horn clause, which is denoted by $\Box$. A many valued Horn formula containing the empty clause is denoted by $\Gamma_\Box$.*

Theorem 3.3.1 Soundness $\Gamma \vdash_{MPUR} \Gamma' \Rightarrow \Gamma \models \Gamma'$

Proof: Since the empty many valued Horn clause is by definition unsatisfiable and it is obtained by a finite number of applications of MPUR rule, it suffices to show that if there exists an interpretation that satisfies both $((\neg p \vee D^-); \alpha)$ and $((p); \alpha')$ provided that $\alpha + \alpha' > 1$; then, this interpretation satisfies $((D^-); \alpha)$.

Assume that $((\neg p \vee D^-); \alpha)$ and $((p); \alpha')$ are satisfiable. Let I be an interpretation that satisfies both $((\neg p \vee D^-); \alpha)$ and $((p); \alpha')$. So, it must be that $I(p) \geq \alpha'$ and $I(\neg p) \leq 1 - \alpha'$. Since $I(\neg p) \leq 1 - \alpha'$ and $1 - \alpha' < \alpha$, the interpretation I satisfies $((D^-); \alpha)$.

Theorem 3.3.2 Completeness Γ *is unsatisfiable iff* $\Gamma \vdash \Gamma_\Box$

The proof is given in [33].

Definition 3.3.6 *A refutation proof of a many valued Horn formula* Γ *denoted by* $\Gamma \vdash_{MPUR} \Box$ *is a finite succession of many value Horn clauses* $\mathcal{C}_1, \ldots, \mathcal{C}_m$ *such that* $\mathcal{C}_m = \Box$ *and, for each* $k(1 \leq k \leq m)$*, either* $\mathcal{C}_k$ *is a clause of* Γ *or* $\mathcal{C}_k$ *is obtained from* $\mathcal{C}_i$ *and* $\mathcal{C}_j(k \geq i, j)$ *applying the MPUR rule.*

3.3.3 MPUR-PROP algorithm

A polynomial algorithm can be obtained directly applying the MPUR rule: the input of the algorithm is a many-valued Γ-formula whose (un)satisfiability will be proved by successive applications of the MPUR inference rule. This rule is applied in a forward chaining strategy until one of the following two cases arises: either the empty clause is derived meaning that the original formula is unsatisfiable, or no more inference rules can be applied which means that the formula Γ is satisfiable. A first version of the algorithm MPUR-PROP(Γ) which propagates the MPUR inference rule is the following:

```
MPUR-PROP(Γ)
  If $\Gamma^+ = \{\}$ then return(sat)
  If $\Box \in \Gamma$ then return(unsat)
  If $(p; \alpha) \in \Gamma$ then
      return(MPUR-PROP(MPUR Γ $(p; \alpha)$))
  return (sat)
```

where:

(MPUR Γ $(p; \alpha)$**):** It applies the MPUR rule returning the formula Γ' resulting of removing the unit clause $(p; \alpha)$, and all the occurrences of the literals $\neg p$ in clauses $((\neg p \vee D^- \vee q); \alpha')$ such that $\alpha + \alpha' > 1$.

It can be remarked that the number of recursive calls to the main procedure is in $O(size(\Gamma))$. Also, the complexity of each execution of MPUR is in $O(size(\Gamma))$ since the blind search of clauses involved in a MPUR step, namely clauses having a $\neg p$ occurrence, requires $O(size(\Gamma))$ time. Hence, the worst-case complexity of the above presented algorithm is in $O(n^2)$.

A more efficient algorithm. Improving algorithm efficiency requires to integrate optimizations as we describe below.

Searching for $\neg p$ occurrences. The aim is to avoid the exhaustive search for clauses containing the literals $\neg p$ after a given deduction $(p; \alpha)$ is made. To this end, we apply a formula pre-process in which a set of pointers $Neg(p)$ for each $p \in Prop(\Gamma)$ is obtained. These pointers are links between propositions p and the clauses containing a literal $\neg p$. Thus, these pointers, allow to access directly the clauses involved in a MPUR procedure avoiding the blind search for such clauses.

Removals of literals. The MPUR procedure removes certain occurrences of literals. From an algorithmic point of view, each removal is in $O(\mid D \mid)$, where $\mid D \mid$ is the cardinal of a disjunction D of a clause $C = ((D^- \vee q); \alpha)$. But with the help of a counter $(Neg.Counter(C))$ the complexity of a removal literal can be reduced to $O(1)$ as indicated below.

Remember that a negative literal $\neg p$ in the part D^- of a clause $C = (D^- \vee q; \alpha)$ is removed when a many valued literal $(p; \alpha')$ is deduced and $\alpha + \alpha' > 1$. Now, these removals will be substituted by decrements of the counter $Counter(C)$. This way, $Counter(C)$ indicates the number of negative literals remaining in D^- not having been affected by the previous many valued literal deductions.

Thus, although no information about which negative literals have been removed from C is stocked, the necessary information of how many negative literals there are left in C is furnished by the counter at each moment of the inference process. When a counter $Neg.Counter(C)$ reaches the zero value, it means that the initial clause $C = ((D^- \vee q); \alpha)$, owing to the virtual removals, has become a positive clause $C = (q; \alpha)$, namely, it is generated a new unit clause. If the atom q does not exists, then it means that the original Γ formula is unsatisfiable.

Treatment of MPUR sequences In the sequential applications of the MPUR rule, a same proposition with different truth value can be deduced. For instance, assume that first $(p; \alpha)$ is deduced and later is deduced $(p; \alpha')$ with $\alpha' < \alpha$. It can be easily checked that the last deduction will produce an MPUR that will simplify the formula. Indeed, the sooner $(p; \alpha)$ deduction removed all the information that could be removed by the later $(p; \alpha')$ deduction. However if $\alpha' > \alpha$ new virtual removals can be performed. To apply the MPUR inferences adequately we stock the maximal truth degree α with which p has been deduced so far, in a data structure $Val(p)$. Thus in the previous cases, the operations for the MPUR inference are launched only when $(p; \alpha')$ is deduced and $\alpha' > Val(p)$. If this is the case $Val(p)$ will be updated with $Val(p) \leftarrow \alpha'$.

Finally, we store the deduced unit clauses $(p; \alpha)$ in a Stack data structure as the basis for an iterative bottom-up algorithmic process.

Henceforth, [X] denotes a pointer to the object X. For example $[C]$ is a pointer to the clause C.

Given the previous data structures and algorithmic operations, the main procedure remains as follows:

```
**MPUR-PROP($\Gamma$)**
while $Stack \neq \emptyset$ do:
  $(p;\alpha) \leftarrow pop(Stack)$
  if $Val(p) < \alpha$ do:
    $Val(p) \leftarrow \alpha$
    for $\forall [C] \in$Neg$(p)$ do:
      if $Val(C) + \alpha > 1$ then
      Decrement $Neg.Counter(C)$
      if $Neg.Counter(C) = 0$ then do:
        if $C^+ = \{\} \in C$ then return 'Unsat'
        Else $(C^+ = q)$ $push((q;Val(C)), Stack)$
return 'Sat'
```

Main procedure complexity. The algorithm is of course more efficient than the previous one. But nevertheless, the complexity is still quadratic.

Theorem 3.3.3 *The complexity of the main procedure is in $O(k \cdot m)$, where k is the maximum number of clauses concluding a same non-negated proposition p and m is the maximum number of clauses sharing the same negated proposition $\neg p$.*

Proof: For each new unit clause $(p;\alpha')$ deduced, the set $Neg(p)$ is scanned. This search is done at most as many times as occurrences of p exist in Γ. So, the computational cost is in $O(k \cdot m)$.

The aim of the following optimization is to design a strictly linear Main procedure. The non-linear complexity factor will be confined to only the Pre-process step.

Ordering $Neg(p)$. Once the $Neg(p)$ lists are obtained, we sort them as follow $Neg(p) = \{[C_1 = (S_1;\alpha_1)], \ldots, [C_k = (S_k;\alpha_k)]\}$ with $\alpha_1 \geq \ldots \geq \alpha_k$. This is done with a call to the well known procedure MergeSort [41], namely $Neg(p) \leftarrow MergeSort(Neg(p))$. Once the $Neg(p)$ lists are obtained, the removals can be performed in a more efficient way.

When a many-valued literal (p,α) is deduced, the first pointer $[C_1]$ to a clause (S_1,α_1) in $Neg(p)$ is considered checking whether $\alpha + \alpha_1 > 1$. In the affirmative case, the pointer is removed from $Neg(p)$, the counter decrements are executed and the same check is carried out with the second clause pointer in $Neg(p)$. These operations are repeated till a certain check is negative and at that moment, the removal of pointers from $Neg(p)$ is stopped. This process ensures that the list $Neg(p)$ is revised at most once.

The definitive algorithm is given below. We first describe the initialization procedure and afterwards the main procedure:

```
**Preprocess-MPUR-PROP($\Gamma$)**
$Stack \leftarrow \emptyset$
for $\forall p \in$ Prop($\Gamma$) do:   Val($p$) $\leftarrow 0$, Neg($p$) $\leftarrow \{\}$
for $\forall C \in \Gamma$ do:
    if $C = \Box$ then return('unsatisfiable')
    else $C = ((\neg p_1 \vee \ldots \vee \neg p_k \vee C^+); \alpha)$ do:
      if $k \neq 0$ then do:
        $Neg.Counter(C) \leftarrow k$
        for $1 \leq i \leq k$ do: Add $[C]$ to $Neg(p_i)$
      else do: $push((p; Val(C)), Stack)$
for $\forall p \in Prop(\Gamma)$ do:
    $Neg(p) \leftarrow MergeSort(Neg(p))$
Main-MPUR-PROP
End
```

```
**Main-MPUR-PROP**
while $Stack \neq \emptyset$ do:
   $(p; \alpha) \leftarrow pop(Stack)$          {PROCEDURE MPUR}
   if $\alpha > Val(p)$ then do:
     $Val(p) \leftarrow \alpha$
     while $Val(First.clause(Neg(p))) + \alpha > 1$ do:
       Remove $First.clause(Neg(p))$ from Neg(p)
       Decrement $Neg.Counter(C)$
       if  $Neg.Counter(C) = 0$ then do:
         if $q = \{\} \in C$ then return 'Unsat'
         else do: $push((q; Val(C)), Stack)$
return 'Sat'
End
```

Theorem 3.3.4 *The complexity of the Preprocess procedure is in $O(n.log(m))$, where n is the number of different propositions and m is the maximal number of negative occurrences of a same proposition.*

Proof: The cost of the first "for" loop is trivially in $O(p)$, where p is the number of propositions in Γ. The second *for* instruction is in $O(n)$, where n is the size of Γ. This is because there is only one iteration for each clause and in such iteration the literals of the clause are scanned only once. As the complexity of MergeSort is known to be in $O(n.log(m))$ in the worst case, so is the complexity of the last line, where n is the number of different propositions and m is the maximal number of negative occurrences of a same proposition. Consequently, the final pre-process complexity is also in $O(n.log(n))$.

Theorem 3.3.5 *The complexity of the Main procedure is in $O(n)$.*

The proof follows from the previous explained optimizations of the steps of this procedure.

Thus, it could be seen that the main procedure is strictly linear and that the non-linear factor has been confined to the preprocess step.

3.4 Regular Horn-CNF SAT problem

3.4.1 Syntax and semantic

The following definitions describe the syntax and semantics of the regular HORN-CNF formulas. A more detailed description about these concepts can be found in [9, 33, 46, 48, 12].

Definition 3.4.1 Signed formulas *Let $N = \{i_1, i_2, \ldots, i_n\}$ be a finite set of truth values, S a subset of N ($S \subseteq N$) and p a proposition. A total order $\leq$ is associated with N. An expression of the form $S\!:\!p$ is a signed literal and S is its sign. Given a signed literal $S\!:\!p$ and a set of truth values N, $(N \setminus S)\!:\!p$ denotes the complement of $S : p$. A signed clause is a disjunction of signed literals. A signed formula is a conjunction of signed clauses.*

Definition 3.4.2 Interpretation and satisfiability *An interpretation $\mathbf{I}$ is a mapping that assigns to every proposition a value in the set of truth values N. An interpretation $\mathbf{I}$ satisfies a signed literal $S\!:\!p$ iff $\mathbf{I}(p) \in S$. An interpretation $\mathbf{I}$ satisfies a signed clause iff $\mathbf{I}$ satisfies at least one of its signed literals. A signed formula Γ is satisfiable iff there exists at least one interpretation that satisfies all the signed clauses in Γ. A signed formula that is not satisfiable is unsatisfiable. The empty signed clause $\Box$ is unsatisfiable and the empty signed formula $\Gamma = \{\}$ is satisfiable.*

Definition 3.4.3 Regular sign *Let $\uparrow i$ denote the set $\{j \in N \,|\, j \geq i\}$ and $\downarrow i$ the set $\{j \in N \,|\, j \leq i\}$, where N is the set of truth values, $\leq$ and $\geq$ are linear orders on N and $i \in N$. If a sign S is equal to either $\downarrow i$ or $\uparrow i$, then it is a regular sign. A signed literal $S\!:\!p$ has positive (resp. negative) polarity if $S = \uparrow i$ (resp. $S = \downarrow i$).*

Definition 3.4.4 Regular formulas *Let R be a regular sign. A regular literal is a signed literal whose sign is regular. A regular clause $\mathcal{C}$ is a disjunction of regular literals $\mathcal{C} = R_1 : p_1 \vee R_2 : p_2 \vee \ldots \vee R_m : p_m$. A regular Horn clause is a regular clause with at most one regular literal with positive polarity. A regular unit clause is a regular clause containing only one regular literal. A regular Horn formula is a conjunction of regular Horn clauses.*

Example 3.4.1 *The following formula is an unsatisfiable regular Horn-CNF formula:*

$$\begin{aligned}
\Gamma = \{&\mathcal{C}_1 = (\uparrow 0.7 : p_1),\\
&\mathcal{C}_2 = (\uparrow 0.6 : p_3),\\
&\mathcal{C}_3 = (\uparrow 0.8 : p_6),\\
&\mathcal{C}_4 = ((\downarrow 0.2 : p_1 \vee \downarrow 0.1 : p_2 \vee \downarrow 0.15 : p_3) \vee\\
&\qquad (\downarrow 0.25 : p_4 \vee \downarrow 0.4 : p_5 \vee \downarrow 0.2 : p_6) \vee\\
&\qquad (\uparrow 0.8 : p_7)),\\
&\mathcal{C}_5 = (\downarrow 0.1 : p_8)\}
\end{aligned}$$

3.4.2 Logical Calculus and an almost linear algorithm

The Logical Calculus and the associated SAT algorithm are similar to those of the previous section. In fact, the Regular Logic is a simple extension of the multi-valued logic of the previous section. Indeed, a clause $C = ((\neg p_1 \vee p_2 \vee \ldots \vee \neg p_k \vee p); \alpha)$ in the previous multi-valued language can be represented in the regular logic by the clause: $(\downarrow 1-\alpha : p_1 \vee 1-\alpha : p_2 \vee \ldots \vee \downarrow 1-\alpha : p_k \vee 1-\alpha : p)$ However, a regular clause $(\downarrow i_1 : p_1 \vee \downarrow i_2 : p_2 \vee \ldots \vee \downarrow i_k : p_k \vee \uparrow i : p)$ can not be represented in the previous multi-valued language.

The Logical Calculus is formed by only one rule that we call the Regular Unit Resolution (RUR).

Definition 3.4.5

$$\frac{(\uparrow j : p), (\downarrow i : p \vee D)(j > i)}{(D)} (RUR)$$

Obviously, when D is empty the empty clause $\Box$ is deduced.

Theorem 3.4.1 correctness $\Gamma \vdash_{RUR} \Box$ *iff* Γ *is unsatisfiable.*

The proof is in [48].

As the inference rule and multivalued logic of the previous section and the current one are similar, the corresponding SAT algorithm is similar to that of the previous section.

```
**Preprocess-RUR-PROP($\Gamma$)**
$Stack \leftarrow \emptyset$
for $\forall p \in \text{Prop}(\Gamma)$ do:   $\text{Val}(p) \leftarrow 0$, $\text{Neg}(p) \leftarrow \{\}$
for $\forall C \in \Gamma$ do:
    if $C = \Box$ then return('unsatisfiable')
    else $(\downarrow i_1 : p_1 \vee \downarrow i_2 : p_2 \vee \ldots \vee \downarrow i_k : p_k \vee \uparrow i : p)$ do:
        if $k \neq 0$ then do:
            $Neg.Counter(C) \leftarrow k$
            for $1 \leq j \leq k$ do: Add $([C]; i_j)$ to $Neg(p_j)$
        else do: $push((\uparrow i : p), Stack)$
for $\forall p \in Prop(\Gamma)$ do:
    $Neg(p) \leftarrow MergeSort(Neg(p))$
Main-MPUR-PROP
End
```

```
**Main-RUR-PROP**
while $Stack \neq \emptyset$ do:
  $(\uparrow i : p) \leftarrow pop(Stack)$
  if $i > Val(p)$ then do:
    $Val(p) \leftarrow i$
    while $Val(First.clause(Neg(p))) < i$ do:
      Remove $First.clause(Neg(p))$ from Neg(p)
      Decrement $Neg.Counter(C)$
      if  $Neg.Counter(C) = 0$ then do:
        if $q = \{\} \in C$ then return 'Unsat'
        else do: $push((\uparrow i : p), Stack)$
return 'Sat'
End
```

The current algorithm is almost linear. This statement derives from the similarity between the previous algorithm in precedent section and the current one. The proof is very similar to those of the previous theorems 3.3.4 and 3.3.5.

Chapter 4

The Horn-NNF propositional SAT Problem

4.1 Introduction

In some practical applications of Computer Science, the well-known Normal Forms CNF and DNF do not provide a natural framework to represent knowledge and reason. In fact, performing inferences efficiently with formulas whose syntactic forms are non restricted to the well-known ones is a matter of major interest in many practical applications inside very heterogeneous areas such as Expert Systems, Deductive Data Bases, Hardware Design, Automated Software Verification, Symbolic Optimization, Logic Programming, Automated Theorem Proving, Petri Nets, Truth Maintenance Systems, etc. For example, in Rule Based Systems, it is interesting to allow rules with a more general syntactic form than the standard $A_1 \wedge A_2 \wedge \ldots A_n \rightarrow B$. Indeed, for example, rules of kind $(((A_1 \vee A_2) \wedge A_3) \vee A_4) \wedge A_5 \wedge A_6 \rightarrow B_1 \wedge B_2$ could be allowed.

However, most of the existing efficient proof methods are designed to work with CNF formulas. So, it is a common practice to translate knowledge representations from general forms to CNF's [26, 80, 55, 71]. This transformation was originally proposed in 1970 by Tseitin [84] who published the first algorithm, later [51] included the case for first-order logic, [68] covered the cases for modal and intuitionist logics, finally, Hähnle [47] investigated the problem of translating arbitrary finitely valued logics to short CNF signed formulas.

Currently, two kind of CNF translations are known, one preserves the logical equivalence and the other only the satisfiability equivalence.

1. In the first case, the translation cannot skip the explosion of the number of symbols due to the $\wedge/\vee$ distribution operation and thus the size of the resulting CNF formula can increase exponentially.

2. The other approach consists in modifying the formula by introducing artificial literals [84] aiming at preserving the satisfiability relation. This

second line of solution has two strong drawbacks. First, the logical equivalence relation is lost which could be invalid for certain applications. For example, in Expert Systems each proposition has a practical and semantical meaning which is vanished if the original propositions are substituted by artificial ones. Second, to solve the SAT problem two procedures are required: the first one transforms the original formula into a CNF formula, and the second one, taking as its input the translated CNF formula which is bigger than the original one, applies properly the satisfiability test.

Hence, we claim that processing directly the non-clausal formula in an appropriate way arises as the most efficient approach to solve non-clausal SAT problems.

In spite of the large number of potential applications, few studies have been devoted to non-clausal reasoning. Thus, we present new results related to this field and more precisely to tractable methods for reasoning with formulas in Negation Normal Form (NNF). More specifically, the main result of this chapter is twofold. Firstly, we identify Negation Normal Form (NNF) formulas Γ having a Horn-like structure and second, and more interestingly, we prove that their associated SAT problem is strictly linear.

The NNF generalization of the well-known Horn CNF formulas

$$\Gamma = \Gamma_1 \wedge \Gamma_2 \wedge \ldots \wedge \Gamma_n$$

where clauses have the syntax:

$$\Gamma_i = \neg p_{i,1} \vee \neg p_{i,2} \vee \ldots \vee \neg p_{i,n(i)} \vee p_i$$

or, in an analogous notation,

$$\Gamma_i = p^-_{i,1} \vee p^-_{i,2} \vee \ldots \vee p^-_{i,n(i)} \vee p^+_i,$$

is

$$\Gamma_i = NNF^-_{i,1} \vee NNF^-_{i,2} \vee \ldots \vee NNF^-_{i,n(i)} \vee C^+_i,$$

where terms $NNF^-_{i,j}$ are general NNF formulas formed exclusively by negated propositions (negative literals). The term C^+_i is a conjunction composed exclusively by propositions (positive literals). These restrictions are indicated respectively with the symbols "-" and "+" in the exponent. We call the associated SAT problem, namely knowing whether Horn-like non-clausal formulas of kind Γ are satisfiable, the HORN-NNF-SAT problem.

The kind of Horn-like NNF (HORN-NNF) formulas we dealt with in this chapter, can arise from an original non-clausal representation of the problem, or if the problem is modelled by a classical Horn formula relied on Knowledge Rule Based Systems, as a result of a factorization operation of its Horn rules. The Horn-like NNF formulas are compact representations of Horn formulas given that they require less symbols than Horn formulas to codify identical problems; this reduction can be in an exponential rate. For instance the HORN-NNF clause:

$$
\begin{gathered}
(\neg p_{1,1} \vee \neg p_{1,2} \vee \ldots \vee \neg p_{1,n}) \\
\wedge \\
(\neg p_{2,1} \vee \neg p_{2,2} \vee \ldots \vee \neg p_{2,n}) \\
\wedge \\
\ldots \\
\wedge \\
(\neg p_{k,1} \vee \neg p_{k,2} \vee \ldots \vee \neg p_{k,n}) \\
\vee \\
(p_1 \wedge p_2 \wedge \ldots \wedge p_n)
\end{gathered}
$$

is equivalent to n^{k+1} Horn clauses.

This chapter is structured as follows. In the next section we briefly review the research already done about tractable satisfiability and related issues. After, we define the SIMPLE-HORN-NNF-SAT problem and the HORN-NNF-SAT problem. The former serves to introduce in a simplified form, the concepts and algorithmic principles required to solve linearly the latter, the general HORN-NNF-SAT problem. Then, for each one of these two SAT problems, we allocate a section composed by the following subsections. The first one defines a sound and refutation complete Logical Calculus. The next one describes the algorithmic schema materializing the Logical Calculus. The third subsection provides an almost linear algorithm following the designed algorithmic schema and finally, in the last subsection, we detail a strictly linear algorithm to resolve the (SIMPLE)-HORN-NNF-SAT problem along with the proofs of its logical correctness and of its strict linearity.

4.2 Related Work

In this section, we briefly review successively the existing computational results of the SAT problem with CNF and NNF formulas.

The propositional satisfiability (CNF-SAT) problem is fundamental at the core in Computer Science. It was the first *NP-complete* problem found [24]. Since then, a rather big effort has been done to determine some CNF-SAT islands of tractability with significant repercussions in applications. The most important classes that can be resolved in deterministic polynomial time are: 2-CNF, for which linear algorithms were designed in [15, 36, 27], and HORN-SAT, that admits also linear algorithms as showed in [28, 38, 40]. In addition to these classes, certain variants of the HORN-SAT problem have been also found out to be solvable in polynomial time.

Several methods have been developed to infer with non-clausal formulas. This is the case of Matings [10], Matrix Connection [17], NC-Resolution [69], Dissolution [70], TAS [1, 42, 73] and polWSAT [83]. Also in [76] a decision procedure for propositional formulas is presented where the original formulas are previously translated into NNF forms. More recently, in [21] a method based on the DPLL [25] algorithm has been designed to deal with non clausal satisfiability. However, no studies relative to NNF tractability employing one of these methods have been carried out.

With some of the previous mentioned methods, the satisfiability of the HORN-NNF formulas can be obtained with a polynomial number of inferences. However, there is not result about the algorithmic complexity. What we claim in this chapter is that the HORN-NNF formulas can be solved in strictly linear algorithmic complexity time which is a much stronger assertion than the inferential polynomiality exhibited by the known methods. Even if the number of inferences of some of the previous mentioned methods was linear this would not imply a linear complexity time. Proving a strictly linear time is a much stronger result than proving a linear number of the inferences. Indeed, in addition to the inferential calculus, we provide an optimized algorithm with appropriately selected data structures. Both elements are essential in the proof of linear time of the computational problem dealt with here.

To our knowledge, the first published results concerning non-clausal tractability comes from [31, 32, 35] where a strictly linear forward chaining algorithm to test for the satisfiability of certain NNF formulas subclass is detailed. Such a class embeds the Horn case as a particular case. In [40] a linear backward algorithm is given for the same NNF subclass of formulas.

New results concerning NNF tractability are reported in [75] where a method called Restricted Fact Propagation is presented which is a quadratic, incomplete non-clausal inference procedure.

More recently, in [81, 82] a significant advance in NNF tractability has been accomplished. The author define a class of formulas by extending the Horn formulas to the field of the NNF formulas. Such extension relies on the concept of polarity. The non-clausal formulas in [81] are somewhat different from the HORN-NNF formulas defined here. A more detailed description of the kind of formulas in [81] goes beyond the subject of this chapter and it can be consulted in such reference. A method to make inferences and potentially to detect refutation formulas is designed. In [81], a SLD-resolution variant with the property of being refutationally complete is showed but its computational complexity is not studied. In [82] a method for propositional Horn-like NNF formulas is described and it is stated that the method is sound, incomplete and linear.

However, concerning the last issue, no algorithm is specified. Indeed the steps of the method are described as different propagations of some truth values in a sparse tree. Then, although it seems that the number of inferences of the proposed method is linear, it is not proved the resulting complexity (w.r.t. the number of computer instructions) of a linear number of truth value propagations on the employed sparse trees.

In previous work carried out by the authors [7, 6], the linearity of some sub-classes of formulas of the general NNF formulas presented here is proved.

4.3 Proof Methodology of the main result

As mentioned in the introduction section, our aim is to prove that the HORN-NNF-SAT problem can be solved in linear time. This proof requires to design a correct algorithm to solve the HORN-NNF-SAT problem with strictly linear complexity. But the design of such kind of optimized algorithms leads to a long and sophisticated algorithm given in section 4.5.4 mainly by two reasons:

- The processed formulas are not in canonical forms, in other words, formulas can have a big nesting degree of the connectors $\wedge/\vee$;
- The optimization of an algorithm requires careful choices of the data structure and the computer operations to be performed.

Therefore, for the case of our algorithm, to prove its correctness w.r.t. the satisfiability test of NNF formulas, directly from the pseudo-code, is an unfeasible goal. Indeed, such proofs turn out to be very long, full of notations and small details and therefore of small readability. Altogether, we can state that checking the correction of such proof will be an arduous task. In other words, the proof of the logical properties, more precisely its SAT correction, of such kind of sophisticated algorithms often are not error-free and however these ones are hard to detect. Thus, this kind of proofs could lead to invalid theorem assertions.

To circumvent this problem, we propose the methodology below. In order to prove **progressively** the logical properties, i.e. the SAT correction, of the final complicated linear algorithm, our methodology is firstly applied to a sub-case of the HORN-NNF-SAT problem, that we call the SIMPLE-HORN-NNF-SAT problem, and afterward it is extended to the general HORN-NNF-SAT problem. So, the whole proof is split into **two phases** and each phase is decomposed in four steps which are briefly described in this section.

4.3.1 Description of the Proof Methodology

The **first step** consists in defining a Sound and Refutation Complete Logical Calculus LC. In the **second one**, we define a SAT algorithm A1 based on the previous Logical Calculus without taking care in its design of the optimization complexity aspects. In the **fourth step**, we specify a strictly linear SAT algorithm A3, and so, A3 is carefully designed. The **third one** is to define an intermediate algorithm A2 whose structural design complexity is between the initial algorithm A1, and the final one A3. Thus, the linear SAT algorithm, namely A3, is the final step of our complexity optimization process, represented by the sequence $< LC, A1, A2, A3 >$.

The following fact gives rise to the mentioned difficulty of the correctness proof of the linear SAT algorithm: *the more optimized is an algorithm, the more complex is its design and the more complicated is its SAT correctness proof.*

To solve this difficulty, the idea behind our approach is, conducting the optimization process verifying that each SAT algorithm in the sequence, is satisfiability equivalent to the previous one in the sequence (The equivalence relationship between LC and $A1$ is to be specified later).

We can describe these steps precisely with a simple First Order Logic formalism. Variables are written in small letters and constants in capital letters. We use four first order predicates:

- SRC, to assert the Soundness and Refutation Completeness of a Logical Calculus x, i.e. $SRC(x)$ is true iff x is a Sound and Refutation Complete Logical Calculus.
- $CORSAT$, to assert the CORrectness of a SAT algorithm x, namely $CORSAT(x)$ stands for: x is a SAT algorithm that running on a formula Γ returns "UNSAT" iff Γ is unsatisfiable
- $SATEQU$, to assert that two SAT algorithms are satisfiability equivalents, i.e. $SATEQU(x_1, x_2)$ says that, for any formula Γ, x_1 is a SAT algorithm which, running on a formula Γ, returns "UNSAT" iff x_2 is also a SAT algorithm which, running on the same formula Γ returns "UNSAT" too.
- $LCEQUSAT(x, y)$, to assert the equivalence between a Logical Calculus and a SAT algorithm with the following meaning: $LCEQUSAT(x, y)$ is true whenever x is a Logical Calculus and y is a SAT algorithm such that, for any formula Γ, x derives $\Box$ from Γ iff y running on Γ returns "UNSAT".

More formally, noting:

$$UNSAT(\Gamma) \equiv \Gamma \text{ is unsatisfiable}$$

and $x(\Gamma)$ the value returned by algorithm x running on formula Γ, the previous logical predicates are precisely defined respectively by the following equivalences:

$$\forall x, SRC(x) \Longleftrightarrow \forall \Gamma, \Gamma \vdash_x \Box \Leftrightarrow UNSAT(\Gamma)$$

$$\forall x, CORSAT(x) \Longleftrightarrow \forall \Gamma, x(\Gamma) = \text{“}UNSAT'' \Leftrightarrow UNSAT(\Gamma)$$

$$\forall x, y, SATEQU(x, y) \Longleftrightarrow \forall \Gamma, x(\Gamma) = y(\Gamma)$$

$$\forall x, y, LCEQUSAT(x, y) \Longleftrightarrow \forall \Gamma, \Gamma \vdash_x \Box \Leftrightarrow y(\Gamma) ='' UNSAT''$$

With this notation, the following corollaries are straightforward:

Corollary 4.3.1

$$\forall x, y, LCEQUSAT(x, y) \wedge SRC(x) \Longrightarrow CORSAT(y)$$

Proof. It follows from the definitions of the predicates in the statements. ■

Corollary 4.3.2

$$\forall x, y, SATEQU(x, y) \wedge CORSAT(x) \Longrightarrow CORSAT(y)$$

Proof. It follows from the definitions of the predicates in the statements. ■

As mentioned before, trying to prove the SAT correction of an algorithm with a complex design structure is not the right way to proceed, because the more complex is an algorithm the more difficult is to analyze it.

Thus, in order to prove $CORSAT(A_{i+1})$ directly and exclusively from its structural design, we prove $CORSAT(A_i) \wedge SATEQU(A_i, A_{i+1})$, that together with the previous Corollary, implies $CORSAT(A_{i+1})$. Thus, transferring this approach throughout the optimization sequence $< LC, A1, A2, A3 >$, we have that the difficulty of proving CORSAT(A3) has been reduced to the simpler proofs SRC(LC), LCEQUSAT(LC,A1), SATEQU(A1,A2) and SATEQU(A2,A3).

- The proof of SRC(LC) is a quite standard proof and it is based on known techniques.
- The remaining three proofs are simple ones because we can choice an algorithm whose structural design is close to its predecessor object in the sequential optimization process. Thus, making a straight parallelism between the instructions of the two algorithms, we can state their SAT equivalence.

Thus, in order to prove the correctness of a linear (SIMPLE)-HORN-NNF-SAT algorithm, we prove successively:

1. The existence of a Sound and Refutation Complete Logical Calculus (LC). This proof will be done via well-known mathematical techniques.
2. The existence of a correct SAT algorithm $SATALG_P$ that is a direct implementation of the previous Logical Calculus LC and whose complexity is polynomial.
3. The existence of a correct SAT algorithm $SATALG_Q$ with quadratic complexity issued from $SATALG_P$ via an optimization step.
4. The existence of a SAT correct algorithm $SATALG_L$ with a strictly linear complexity issued from $SATALG_Q$ via an optimization step.

4.3.2 Main Theorems of the Proof Methodology

We distinguish four MAIN THEOREMS in our proof methodology, one for each step of the proof of linearity of the (SIMPLE-)HORN-NNF SAT problem. They are stated below together with a sketch of their proof. In addition to the previous First Order Predicates, we also use $PSAT(x), QSAT(x)$ and $LSAT(x)$ which state that x is a SAT algorithm of respectively Polynomial, Quadratic and Linear complexity.

1. **Logical Calculus (LC)** This step consists in establishing a Logical Calculus appropriate to solve the (SIMPLE)-HORN-NNF-SAT problem. Thus, we must prove its Soundness and Refutation Completeness.

   MAIN THEOREM 1: $\exists x, SRC(x)$

   Proof Sketch. The proof consists in finding a particular Logical Calculus LC and proving its Soundness and Refutation Completeness. As it will be shown, this proof follows standard mathematical techniques. ■

2. **Algorithmic Version of the Logical Calculus** This step consists in defining an Algorithmic Version of the Logical Calculus (AVLC), proving its correctness, w.r.t. the satisfiability test, and its polynomial complexity.

   MAIN THEOREM 2: $\exists x, CORSAT(x) \wedge PSAT(x)$

   Proof Sketch. The proof is based on the first corollary and it is divided in two steps:

   (a) Firstly, we choose a particular $SATALG_P$ for x, namely a polynomial SAT algorithm such that the proof of $LCEQUSAT(LC, SATALG_P)$ is straightforward. Namely, our proposed $SATALG_P$ is a direct algorithmic materialization of the concrete LC defined in the first theorem. Then, applying corollary 1, we deduce $CORSAT(SATALG_P)$.

   (b) Second, the proof of $PSAT(SATALG_P)$ follows from a very simple complexity analyze of the designed $SATALG_P$.

   ■

3. **Quadratic Algorithm (QA)** This step consists in modelling the variables of the previous $SATALG_P$ with specific data structures, and in proving that there exists a Quadratic Algorithm $SATALG_Q$ derived from $SATALG_P$.

   MAIN THEOREM 3 $\exists x, CORSAT(x) \wedge QSAT(x)$

   Proof Sketch. It is based on corollary 4.3.2. We show that there exists a quadratic algorithm $SATALG_Q$ very similar to the previous $SATALG_P$. Given such similarity, we prove easily the statement $SATEQU(SATALG_P, SATALG_Q)$. Thus, $CORSAT(SATALG_Q)$ follows from $SATEQU(SATALG_P, SATALG_Q)$, $CORSAT(SATALG_P)$, proved in the previous theorem, and the corollary 2. $QSAT(SATALG_Q)$ is obtained by an easy complexity analyze. ■

4. **Linear Algorithm (LA)** Finally, the optimal algorithm is described progressively by choosing appropriately the data structures and the computer instructions, aiming at the design of a strictly linear algorithm. The substitutions of the initial data structure and the initial computer instructions by the new ones are explained accurately.

   MAIN THEOREM 4 $\exists x, CORSAT(x) \wedge LSAT(x)$

   Proof Sketch. $SATEQU(SATALG_Q, SATALG_L)$ follows from the tight similarity between $SATALG_Q$ and $SATALG_L$. $CORSAT(SATALG_L)$ is straightforwardly derived from such SAT equivalency, from $CORSAT(SATALG_Q)$, proved in the previous theorem, and from the corollary 4.3.2. ■

The advantage of proposed methodology is that the difficulty of obtaining a proof of the logical properties of the final complex algorithm is simplified, substituting the original proof by the proof of the Soundness and Refutation Completeness of a Logical Calculus, which is done as mentioned by standard formal techniques. The other three SAT equivalence proofs are trivial. In that way, we eliminate many possible sources of error when proving our main result, namely that (SIMPLE-)HORN-NNF-SAT is strictly linear and hence, by showing that there exists a correct and strictly linear algorithm to solve it.

4.4 The SIMPLE-HORN-NNF-SAT problem is linear

In this section, we will develop the four steps of the methodology previously described to obtain an algorithm with strict linear worst-case complexity for the SIMPLE-HORN-NNF-SAT problem.

4.4.1 The SIMPLE-HORN-NNF formulas

Firstly, we recall the required definitions of classical logic and afterward, we introduce the class of HORN-NNF formulas dealt with here and then, we define their associated SAT problem.

Definition 4.4.1 *A literal L is either an atomic proposition $p \in \mathbf{P}$, or its negation $\neg p$. A (negated) proposition (resp. negated proposition) is called positive (resp. negative) literal. A* classical clause *is a finite disjunction of literals: $(L_1 \vee \ldots \vee L_k)$. A unit clause (L) includes only one literal. We denote the empty clause $()$ by $\square$. A* Horn clause *is a classical clause with at most one positive literal. A* Horn formula *is a conjunction of Horn clauses.*

Definition 4.4.2 *D^- (resp. C^-) stands for a disjunction (resp. conjunction) of negative literals, namely $D^- = (\neg p_1 \vee \neg p_2 \ldots \vee \neg p_k)$ (resp. $C^- = \{\neg p_1 \wedge$*

$\neg p_2 \wedge \ldots \wedge \neg p_k\}$. $C^+ = \{p_1 \wedge p_2 \wedge \ldots \wedge p_k\}$ stands for a conjunction formed exclusively by positive literals.

Definition 4.4.3 *A CNF^- (resp. DNF^-) formula is a CNF (resp. DNF) formula formed exclusively by negative literals. We write them respectively:*

$$\{D_1^- \wedge D_2^- \wedge \ldots \wedge D_k^-\}$$

$$(C_1^- \vee C_2^- \vee \ldots \vee C_n^-)$$

Remark. Whenever there will not be confusion, we will not use the '()' to begin and to end a disjunction, and we will write:

$$C_1^- \vee C_2^- \vee \ldots \vee C_n^-$$

Definition 4.4.4 *A SIMPLE-HORN-NNF sub-formula $\mathcal{F}$ is a positive unit sub-formula (p), or a disjunction of three optional terms, noted $\mathcal{F} = (DNF^- \vee CNF^- \vee C^+)$. Sub-formulas without (resp. with) the C^+ term are said negative (resp. non-negative) sub-formulas.*

Example 4.4.1 *The following sub-formula is an example of a SIMPLE-HORN-NNF sub-formula:*

$$(\{\neg p_1 \wedge \neg p_2\} \vee \{\neg p_3\} \vee \{(\neg p_4 \vee \neg p_5) \wedge (\neg p_6)\} \vee \{p_7 \wedge p_8\})$$

where the three terms DNF^-, CNF^- and C^+, can be easily identified:

$$CNF^- = \{(\neg p_4 \vee \neg p_5) \wedge (\neg p_6)\}$$

$$DNF^- = \{\neg p_1 \wedge \neg p_2\} \vee \{\neg p_3\}$$

$$C^+ = \{p_7 \wedge p_8\}$$

Definition 4.4.5 *A SIMPLE-HORN-NNF formula Γ is a finite conjunction of SIMPLE-HORN-NNF sub-formulas Γ_i. We note $\Gamma_\Box$ any formula containing the empty clause $\Box$.*

Example 4.4.2 *An example of this kind of formulas is (the fourth sub-formula is the sub-formula of the previous example):*

$$F = \{(p_1) \wedge (p_3) \wedge (p_6)$$

$$\wedge$$

$$(\{\neg p_1 \wedge \neg p_2\} \vee \{\neg p_3\} \vee \{(\neg p_4 \vee \neg p_5) \wedge (\neg p_6)\} \vee \{p_7 \wedge p_8\})$$

$$\wedge$$

$$(\{(\neg p_1 \vee \neg p_7) \wedge (\neg p_9)\} \vee \{p_8 \wedge p_9\})$$

$$\wedge$$

$$\begin{array}{ccccccc}
\neg p_1 & & & & \neg p_4 \vee \neg p_5 & & p_7 \\
\wedge & \vee & \neg p_3 & \vee & \wedge & \vee & \wedge \\
\neg p_2 & & & & \neg p_6 & & p_8
\end{array}$$

Figure 4.1: Bidimensional representation of the fourth clause

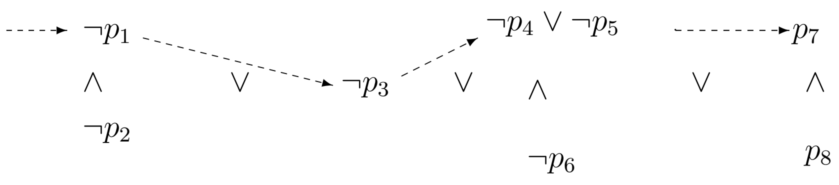


Figure 4.2: A formula path

$$(\{\neg p_8\})\}$$

The fourth (non-unitary) sub-formula is equivalent to the following eight clauses:

$$(\neg p_1 \vee \neg p_3 \vee \neg p_4 \vee \neg p_5 \vee p_7)$$

$$(\neg p_1 \vee \neg p_3 \vee \neg p_4 \vee \neg p_5 \vee p_8)$$

$$(\neg p_1 \vee \neg p_3 \vee \neg p_6 \vee p_7)$$

$$(\neg p_1 \vee \neg p_3 \vee \neg p_6 \vee p_8)$$

$$(\neg p_2 \vee \neg p_3 \vee \neg p_4 \vee \neg p_5 \vee p_7)$$

$$(\neg p_2 \vee \neg p_3 \vee \neg p_4 \vee \neg p_5 \vee p_8)$$

$$(\neg p_2 \vee \neg p_3 \vee \neg p_6 \vee p_7)$$

$$(\neg p_2 \vee \neg p_3 \vee \neg p_6 \vee p_8)$$

The CNF clause represented by a SIMPLE-HORN-NNF sub-formula can be obtained by an exhaustive enumerating of the *paths* (defined below) in a bi-dimensional representation of the formula. The fourth sub-formula of Γ can be represented graphically in two dimensions as follows:

Definition 4.4.6 Bi-dimensional Representation *In the graphical representation of a formula, terms in a conjunction (resp. disjunction) are represented vertically (resp. horizontally).*

Definition 4.4.7 *A* formula path *crosses throughout only one (resp. all) literal (resp. literals) that form(s) a conjunction (resp. disjunction).*

Example 4.4.3 *In the previous example there are exactly eight* formula paths.

The set of literals in each classical simple clause represented in a factorized way by a SIMPLE-HORN-NNF clause, can be obtained by taking the literals crossed in a specific *formula path* in the graphical representation corresponding to the SIMPLE-HORN-NNF clause.

Example 4.4.4 *The clause corresponding to the path indicated by a dashed line in the previous figure is:*

$$\neg p_1 \vee \neg p_3 \vee \neg p_4 \vee \neg p_5 \vee p_7$$

Proposition 4.4.1 *There is a bijection between the set of simple clauses represented in a SIMPLE-HORN-NNF clause and the set of paths defined in its bi-dimensional representation.*

The proof is trivial.

Now, we define the classical semantic concepts related to the formula satisfiability problem.

Definition 4.4.8 *An interpretation I assigns to each formula Γ one value in the set $\{0, 1\}$ and it satisfies a:*

- *pos. literal p (resp. neg. literal $\neg p$) iff $I(p) = 1$ ($I(p) = 0$)*
- *pos. conjunction $I(C^+) = I(\{p_1 \wedge p_2 \wedge \ldots \wedge p_k\}) = 1$, iff $\forall p_i, I(p_i) = 1$*
- *neg. conjunction $I(C^-) = I(\{\neg p_1 \wedge \neg p_2 \wedge \ldots \wedge \neg p_k\}) = 1$, iff $\forall p_i, I(p_i) = 0$*
- *neg. disjunction $I(D^-) = I((\neg p_1 \vee \neg p_2 \vee \ldots \vee \neg p_k)) = 1$, iff $\exists p_i, I(p_i) = 0$*
- *CNF^- term $I(CNF^-) = I(\{D_1^- \wedge D_2^- \wedge \ldots \wedge D_k^-\}) = 1$, iff $\forall D_i, I(D_i^-) = 1$*
- *DNF^- term $I(DNF^-) = I((C_1^- \vee C_2^- \vee \ldots \vee C_k^-)) = 1$, iff $\exists C_i^-, I(C_i^-) = 1$*
- *SIMPLE-HORN-NNF clause $I(\mathcal{F}) = I(CNF^- \vee DNF^- \vee C^+)) = 1$, iff $I(DNF^-) = 1$ or $I(CNF^-) = 1$ or $I(C^+) = 1$.*
- *SIMPLE-HORN-NNF formula Γ iff I satisfies all its SIMPLE-HORN-NNF clauses.*

Definition 4.4.9 *An interpretation I is a* model *of a formula Γ if it satisfies the formula. We say that Γ is* satisfiable *if it has at least one model, otherwise, it is* unsatisfiable.

Definition 4.4.10 *By definition, the conjunction $\{()\}$ and the disjunction $() \equiv \Box$ are unsatisfiable. On the other hand, the conjunction $\{\}$ and the disjunction $(\{\})$ are satisfiable.*

Definition 4.4.11 *The SIMPLE-HORN-NNF-SAT problem is the problem of deciding whether a SIMPLE-HORN-NNF formula is satisfiable.*

4.4.2 Logical Calculus

In this sub-section, we are going to prove the first theorem of our proof methodology, namely:

MAIN THEOREM 1: $\exists x, SRC(x)$

The Logical Calculus needed to prove the (un)satisfiability of SIMPLE-HORN-NNF formulas is formed by rules derived from the general NNF resolution [69]. More specifically, the nucleus of our Logical Calculus is a particular case of the Non-clausal Resolution rule by considering that one of the resolved formulas is a unit clause. The relationships between, on the one hand, the standard Resolution rule and the well-known Unit Resolution rule, and on the other hand, between the Non-clausal Resolution and the Unit General Resolution rules proposed here, are alike.

$$\text{RESOLUTION} \Rightarrow_{Horn} \text{UNIT-RESOLUTION}$$

$$\text{NON-CLAUSAL RESOL.} \Rightarrow_{Horn-NNF} \text{GENERAL UNIT-RESOL.}$$

Definition 4.4.12 *The Logical Calculus is formed by four rules. The first one, called $SIMPLIF$, allows to simplify formulas. The second and third rules are two Positive General Unit Resolution rules, called DUR and CUR, for respectively Disjunction Unit Resolution and Conjunction Unit Resolution appropriate for the SIMPLE-HORN-NNF formulas. The last rule, called the And Elimination rule (AE), enables to obtain unit clauses from a positive conjunction (C^+). We recall that both "()" and $\Box$ denote the empty clause.*

$$\frac{(\{\Box \wedge D_2^- \wedge \ldots \wedge D_k^-\} \vee DNF^- \vee C^+)}{(DNF^- \vee C^+)}(SIMPLIF)$$

$$\frac{(p_1), (\{(\neg p_1 \vee \neg p_2 \ldots \vee \neg p_n) \wedge D_2^- \wedge \ldots \wedge D_k^-\} \vee DNF^- \vee C^+), n \geq 1}{(\{(\neg p_2 \vee \ldots \vee \neg p_n) \wedge D_2^- \wedge \ldots \wedge D_k^-\} \vee DNF^- \vee C^+)}(DUR)$$

$$\frac{(p_1), (CNF^- \vee \{\neg p_1 \wedge \neg p_2 \wedge \ldots \neg p_n\} \vee C_2^- \vee \ldots \vee C_k^- \vee C^+), n \geq 1}{(CNF^- \vee C_2^- \vee \ldots \vee C_k^- \vee C^+)}(CUR)$$

$$\frac{(\{p_1 \wedge \ldots \wedge p_i \wedge \ldots \wedge p_n\})}{(p_1) \wedge \ldots \wedge (p_i) \wedge \ldots \wedge (p_n)}(AE)$$

Remark Note that if the DUR rule is applied with n=1 then the deduced term is of kind $(\{\Box \wedge D_2^- \wedge \ldots \wedge D_k^-\} \vee DNF^- \vee C^+)$.

Remark If the formula Γ_i, to be matched with the antecedent of an inference rule in the previous Logical Calculus, has only one or two terms among the three possible terms CNF^-, DNF^- or C^+ of a SIMPLE-HORN-NNF clause, the missing(s) term(s) in the antecedent is (are) missing also in the consequent. For example, assume that $\Gamma_i = (\{\neg p_1 \wedge \neg p_2\} \vee \{p_4\})$, then Γ_i is of kind $(DNF^- \vee C^+)$. As Γ_i has no a CNF^- term, if we apply the CUR rule over Γ_i with unit clause (p_1), the result is $(\{p_4\}) = (C^+)$, i.e. CUR has been applied ignoring the term CNF^- in its antecedent and consequent.

Remark Note that when we have a unit clause (p_1), the computation steps to apply the corresponding inference rules DUR or CUR depend on whether $\neg p_1$ is in a negative disjunction D^- or in a negative conjunction C^-, but they are **analogous** ones. Indeed, whenever $\neg p_1$ is in a disjunction:

$$D^- = (\neg p_1 \vee \neg p_2 \vee \ldots \vee \neg p_n)$$

or in a conjunction:

$$C_1^- = \{\neg p_1 \wedge \neg p_2 \wedge \ldots \wedge \neg p_n\}$$

which in turn, it is in a DNF^- term:

$$DNF^- = C_1^- \vee C_2^- \vee \ldots \vee C_n^-$$

such literal $\neg p_1$ (resp. that conjunction C_1^-) is removed from the disjunction D^- (resp. DNF^-) with rule DUR (resp. CUR). If literal $\neg p_1$ (resp. conjunction C_1^-) was the only one in the original disjunction D^- (resp. term CNF^-), or it was the last remaining literal (resp. conjunction) from the initial disjunction D^-, (resp. term CNF^-) because the other original literals

$$(\neg p_2 \ldots \vee \neg p_n) \text{ (resp. conjunctions } C_2 \vee \ldots \vee C_n)$$

have been removed previously, then $\Box$ is derived. In other words, D^- (resp. DNF^-) is transformed in $\Box$ after the removals of all of its propositions $\neg p_i$ (resp. conjunctions C_i^-).

Definition 4.4.13 Refutation Proof *A* refutation *of a SIMPLE-HORN-NNF formula Γ, is a finite succession of SIMPLE-HORN-NNF sub-formulas*

$$< F_1, F_2, \ldots, F_n >, \text{ ending with } F_n = \Box$$

where F_i, $1 \leq i \leq n$, is a subformula in Γ or it is obtained by applying one inference rule of the Logical Calculus, in previous definition 4.4.12, upon one subformula (SIMPLIF, AE) or two sub-formulas (DUR,CUR) in the set $\{F_1, \ldots, F_{i-1}\}$.

Example 4.4.5 *Let us consider the formula of the previous example:*

$$\Gamma = \{(p_1) \wedge (p_3) \wedge (p_6)$$

$$\wedge$$

$$(\{\neg p_1 \wedge \neg p_2\} \vee \{\neg p_3\} \vee \{(\neg p_4 \vee \neg p_5) \wedge (\neg p_6)\} \vee \{\neg p_7\})$$
$$\wedge$$
$$(\{(\neg p_1 \vee \neg p_7) \wedge (\neg p_9)\} \vee \{\neg p_8 \wedge p_9\})$$
$$\wedge$$
$$(\{\neg p_8\})\}$$

With the following notations:

$$\Gamma_1 = (p_1), \quad \Gamma_2 = (p_3), \quad \Gamma_3 = (p_6),$$

$$\Gamma_4 = (\{\neg p_1 \wedge \neg p_2\} \vee \{\neg p_3\} \vee \{(\neg p_4 \vee \neg p_5) \wedge (\neg p_6)\} \vee \{p_7\}),$$

$$\Gamma_5 = (\{(\neg p_1 \vee \neg p_7) \wedge (\neg p_9)\} \vee \{p_8 \wedge p_9\}),$$

$$\Gamma_6 = (\{\neg p_8\});$$

a proof sequence of the unsatisfiability of this formula Γ *is the following:*

$$\frac{\Gamma_1 = (p_1), \Gamma_4 = (\{\neg p_1 \wedge \neg p_2\} \vee \{\neg p_3\} \vee \{(\neg p_4 \vee \neg p_5) \wedge (\neg p_6)\} \vee \{p_7\})}{\mathcal{F}_7 = (\{\neg p_3\} \vee \{(\neg p_4 \vee \neg p_5) \wedge (\neg p_6)\} \vee \{p_7\})}(CUR)$$

$$\frac{\Gamma_2 = (p_3), \mathcal{F}_7 = (\{\neg p_3\} \vee \{(\neg p_4 \vee \neg p_5) \wedge (\neg p_6)\} \vee \{p_7\})}{\mathcal{F}_8 = (\{(\neg p_4 \vee \neg p_5) \wedge (\neg p_6)\} \vee \{p_7\})}(CUR)$$

$$\frac{\Gamma_3 = (p_6), \mathcal{F}_8 = (\{(\neg p_4 \vee \neg p_5) \wedge (\neg p_6)\} \vee \{p_7\})}{\mathcal{F}_9 = (\{(\neg p_4 \vee \neg p_5) \wedge \Box\} \vee \{p_7\})}(DUR)$$

$$\frac{\mathcal{F}_9 = (\{(\neg p_4 \vee \neg p_5) \wedge \Box\} \vee \{p_7\})}{\mathcal{F}_{10} = (\{p_7\})}(SIMPLIF)$$

$$\frac{\mathcal{F}_{10} = (\{p_7\})}{\mathcal{F}_{11} = (p_7)}(AE)$$

$$\frac{\Gamma_5 = (\{(\neg p_1 \vee \neg p_7) \wedge (\neg p_9)\} \vee \{p_8 \wedge p_9\}), \mathcal{F}_{11} = (p_7)}{\mathcal{F}_{12} = (\{(\neg p_1) \wedge (\neg p_9)\} \vee \{p_8 \wedge p_9\})}(DUR)$$

$$\frac{\mathcal{F}_{12} = (\{(\neg p_1) \wedge (\neg p_9)\} \vee \{p_8 \wedge p_9\}), \Gamma_1 = (p1)}{\mathcal{F}_{13} = (\{\Box \wedge (\neg p_9)\} \vee \{p_8 \wedge p_9\})}(DUR)$$

$$\frac{\mathcal{F}_{13} = (\{\Box \wedge (\neg p_9)\} \vee \{p_8 \wedge p_9\})}{\mathcal{F}_{14} = (\{p_8 \wedge p_9\})}(SIMPLIF)$$

$$\frac{\mathcal{F}_{14} = (\{p_8 \wedge p_9\})}{\mathcal{F}_{15} = (p_8), \mathcal{F}_{16} = (p_9)}(AE)$$

$$\frac{\Gamma_6 = (\{\neg p_8\})}{\mathcal{F}_{17} = (\neg p_8)}(AE)$$

$$\frac{\mathcal{F}_{15} = (p_8), \mathcal{F}_{17} = (\neg p_8)}{\mathcal{F}_{18} = \Box}(CUR)$$

Theorem 4.4.1 Correctness *Let us LC={SIMPL,DUR,CUR,AE}.*

$$SRC(LC)$$

Or in other terms,

$$\Gamma \vdash_{LC} \Box \Longleftrightarrow UNSAT(\Gamma)$$

To prove the previous theorem, we have to prove the Soundness and Refutation Completeness of the defined Logical Calculus.

Lemma 4.4.1 Soundness

$$\Gamma \vdash_{SIMPLIF} \mathcal{F} \Longrightarrow \Gamma \models \mathcal{F}$$

$$\Gamma \vdash_{DUR} \mathcal{F} \Longrightarrow \Gamma \models \mathcal{F}$$

$$\Gamma \vdash_{CUR} \mathcal{F} \Longrightarrow \Gamma \models \mathcal{F}$$

$$\Gamma \vdash_{AE} (p) \Longrightarrow \Gamma \models (p)$$

Proof. The soundness of each rule of the Logical Calculus follows from the definitions of model and that of logical consequence. ■

Lemma 4.4.2 Refutation Completeness

$$UNSAT(\Gamma) \Longrightarrow \Gamma \vdash \Box$$

Proof. Base case

Assume the formula has no literals. The only formulas without literals are:

$$\Gamma = \{\}$$

and

$$\Gamma = \{\Gamma^1_\Box \wedge \Gamma^2_\Box \wedge \ldots \wedge \Gamma^k_\Box\}$$

where each clause $\Gamma^i_\Box$ is formed by the following three optional terms:

$$\Gamma^i_\Box = (\{() \wedge () \ldots \wedge ()\} \vee -\{\} \vee \{\} \vee \ldots \vee \{\} - \vee \{\})$$

Given that the $DNF^- = \{\} \vee \{\} \vee \ldots \vee \{\}$ term and the $C^+ = \{\}$ term are satisfiable ones, the only unsatisfiable formulas are those containing at least one clause $\Gamma^i_\Box$ without anyone of the three terms, $\Gamma^i_\Box = () \equiv \Box$, or only containing the CNF^- term $\Gamma^i_\Box = (\{() \wedge () \ldots \wedge ()\})$. In these both cases, $\Gamma^i_\Box$ is unsatisfiable and hence, the formula Γ is also unsatisfiable.

So, we need to prove the theorem only for these two cases of formulas.

1. Case $\Gamma^i_\Box = \Box$. $\Box$ is deduced by the reflexivity property of the inference relation $\vdash$.

2. Case $\Gamma^i_\Box = (\{() \wedge () \ldots \wedge ()\})$. In this case, applying the $SIMPLIF$ rule, the clause $() \equiv \Box$ is deduced.

Let us prove the theorem for formulas with literals.

We note $\mathcal{F} = (C^-_1 \vee C^-_2 \vee \ldots \vee C^-_k \vee \{D^-_1 \wedge D^-_2 \wedge \ldots \wedge D^-_n\} \vee \mathcal{C}^+)$
Let us $pos(p) = \{\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_k\}$ be the set of clauses s. t. $p \in \mathcal{C}^+_i$, $1 \leq i \leq k$.
We define rank(p) as follows:

If $\mathcal{F} = (C^+)$ and $p \in C^+$ then $rank(p) = 1$.
If $\neg p \in \mathcal{F}$ and $pos(p) = \emptyset$ then $rank(p) = 1$.

$$rank(p) = 1 + max\{rank(p_i) : \neg p_i \in \mathcal{F} \in pos(p)\}$$

We note Γ^+ the set of clauses of Γ containing a positive conjunction C^+ and Γ^- the set of clauses of Γ containing exclusively negated literals.

Lemma 4.4.3 *If $\Gamma^+ \models p$ and $rank(p) = k$ then there exists at least one clause $\mathcal{F} = (C^-_1 \vee C^-_2 \vee \ldots \vee C^-_k \vee \{D^-_1 \wedge D^-_2 \wedge \ldots \wedge D^-_n\} \vee \mathcal{C}^+)$ such that*

1. *$p \in \mathcal{C}^+$, or similarly, $\mathcal{F} \in pos(p)$*

2. *$\forall \neg p_i \in \mathcal{F}$, $rank(p_i) < k$*

3. *$\forall C^-_i$,$\exists \neg p_i$, $\neg p_i \in C^-_i$ s.t. $\Gamma^+ \models p_i$ and $\exists D^-_j$, $\forall \neg p_j$,$\neg p_j \in D^-_j$ s.t. $\Gamma^+ \models p_j$.*

Proof.

1. Assume $\nexists \mathcal{F}, \mathcal{F} \in pos(p)$. Let us I be a model of Γ^+ such that $I(p) = 1$. Let us consider an interpretation I' with the same mapping that I except for $I'(p) = 0$. Obviously I is also a model of Γ^+ because I' satisfies more literals than I since all the occurrences of p in Γ^+ are negated. Hence, $\Gamma^+ \not\models p$.

2. Assume that $\exists \neg p_i$ such that $rank(p_i) = l \geq k$. Then by definition of rank, $rank(p) = l + 1 > k$.

3. Assume that $\exists C_i^-, \forall \neg p_i$, $\neg p_i \in C_i^-$ such that $\Gamma^+ \not\models p_i$ or assume that $\forall D_j^-$, $\exists \neg p_j, \neg p_j \in D_j^-$, $\Gamma^+ \not\models p_j$. We provide the proof for the first alternative, the proof for the second one is completely similar. If p_i is not a logical consequence means that there is a model of Γ^+ such that $I(p_i) = 0$ and then $I(C_i^-) = 1$. But, taking into account this fact, defining I' equal to I except for the mapping $I'(p) = 0$ is also a model and then we have $\Gamma^+ \not\models p$.

■

Lemma 4.4.4 $\Gamma^+ \models p \Longrightarrow \Gamma \vdash (p)$

Proof. By induction of the statement: If propositions of rank < k are deduced then, propositions of rank k are also deduced.

INDUCTION BASE: The statement is true for propositions of rank 1. Obviously if rank(p)=1, by definition, $p \in \mathcal{C}^+ \in \Gamma$. Then applying $AE(\mathcal{C}^+)$ we produce (p) because $p \in \mathcal{C}^+$.

INDUCTION HYPOTHESIS: Assume that the induction statement is true for literals of rank smaller than k and consider that p is of rank k. Then, by previous lemma there exists at least one clause
$\mathcal{F} = (C_1^- \vee C_2^- \vee \ldots \vee C_k^- \vee \{D_1^- \wedge D_2^- \wedge \ldots \wedge D_n^-\} \vee \mathcal{C}^+)$ such that

1. $p \in \mathcal{C}^+$, or similarly, $\mathcal{F} \in pos(p)$

2. $\forall \neg p_i \in \mathcal{F}$, $rank(p_i) \leq k$

3. $\forall C_i^-, \exists \neg p_i$, $\neg p_i \in C_i^-$ s.t. $\Gamma^+ \models p_i$ and $\exists D_j^-, \forall \neg p_j, \neg p_j \in D_j^-$ s.t. $\Gamma^+ \models p_j$.

Then, by induction hypothesis, $\Gamma^+ \vdash (p_i)$ and $\Gamma^+ \vdash (p_j)$. Applying iteratively the CUR rule upon $\mathcal{F}$ and each p_i we obtain $\mathcal{F} = (\{D_1^- \wedge D_2^- \wedge \ldots \wedge D_n^-\} \vee \mathcal{C}^+)$. Next, applying iteratively the DUR rule upon $\mathcal{F}$ and each p_j we obtain $\mathcal{F} = (\{D_1^- \wedge D_2^- \wedge \ldots \wedge D_{j-1}^- \wedge \Box \wedge D_{j+1}^- \wedge \ldots \wedge D_n^-\} \vee \mathcal{C}^+)$. Next, applying SIMPLIF rule upon $\mathcal{F}$, we deduce $(\mathcal{C}^+)$ and followed by an AE rule we derive (p) because $p \in \mathcal{C}^+$. ■

Lemma 4.4.5 *$UNSAT(\Gamma) \Longrightarrow \exists \mathcal{F}^-, \mathcal{F}^- \in \Gamma^-$ such that:*

1. $\mathcal{F}^- = (C_1^- \vee C_2^- \vee \ldots \vee C_k^- \vee \{D_1^- \wedge D_2^- \wedge \ldots \wedge D_n^-\})$

2. $\forall C_i^-$, $\exists \neg p_i$, $\neg p_i \in C_i^-$ s.t. $\Gamma^+ \models p_i$

3. $\exists D_j^-$, $\forall \neg p_j, \neg p_j \in D_j^-$ s.t. $\Gamma^+ \models p_j$

Proof.

1. By definition of Γ^-, $\mathcal{F}^-$ has the correct structure.

2. The proof is by contradiction. Assume $\exists C_i^-, \forall \neg p_i, \neg p_i \in C_i^-$ s.t. $\Gamma^+ \not\models p_i$. Take a model I of Γ^+ such that $I(p_i) = 0$. Such a model exists because by hypothesis $\Gamma^+ \not\models p_i$. We have also $I(\mathcal{F}^-) = 1$ and hence $I(\Gamma^-) = 1$ and as I is a model of Γ^+ it turns out that Γ is satisfiable.

3. Assume $\forall D_j^-$, $\exists \neg p_j, \neg p_j \in D_j^-$ s.t. $\Gamma^+ \not\models p_j$. The proof of this case is similar to the previous case.

■

Lemma 4.4.6 $UNSAT(\Gamma) \Longrightarrow \Gamma \vdash \Box$

Proof. As we have $\Gamma^+ \models p_i$, by the previous lemma, $\Gamma^+ \vdash (p_i)$. Applying iteratively CUR rule upon each p_i and $\mathcal{F}$ we get $\mathcal{F}^- = (\{D_1^- \wedge D_2^- \wedge \ldots \wedge D_n^-\})$. Now applying iteratively DUR upon each p_j and $\mathcal{F}^-$ we get $\mathcal{F}^- = (\{D_1^- \wedge D_2^- \wedge \ldots \wedge D_{j-1}^- \wedge \Box \wedge D_{j+1}^- \wedge \ldots \wedge D_n^-\})$. Applying SIMPLIF upon $\mathcal{F}$ we obtain (), that is logically equivalent by definition to $\Box$. ■

As the previous lemma is the same that the completeness theorem, the proof of this last lemma concludes the proof of the completeness theorem ■

This proves **MAIN THEOREM 1** of our proof methodology.

4.4.3 Logical Calculus: Its polynomial algorithmic version

In this section, we were going to prove MAIN THEOREM 2 of our proof methodology:

MAIN THEOREM 2: $\exists x, CORSAT(x) \wedge PSAT(x)$

A proof of the unsatisfiability of a SIMPLE-HORN-NNF formula is constructed by applying consecutively inferences rules, as it has been indicated in the definition of **Refutation Proof** and in the subsequent example. Applying this process of inference rules sequencing does not lead to an **efficient** SAT algorithm. One of the original causes is because new sub-formulas of the original formula, are generated and then added to the original formula. Thus, the formula obtained after a certain number of inferences is the original formula augmented with copies of some of their sub-formulas. However, it is easy to prove that the **upper-bound complexity** of a SAT algorithm implementing the Logical Calculus is **Polynomial**

Definition 4.4.14 *Let us define $SATALG_P$, an algorithm that implements the defined Logical Calculus as follows:*

1. *It choices the inferences to be applied according to a deterministic criterion*

2. *It returns "UNSAT" iff the defined Logical Calculus derives $\Box$*

More specifically, the algorithm scheme is the following, where Γ is the initial set of subformulas.

```
$SATALG_P(\Gamma)$
    While the set of executables inference rules is not empty do:
      $\forall (C^+) \in \Gamma$ do: $\forall p \in (C^+) \in \Gamma$: add $(p)$ to $\Gamma$
      Execute applicable DUR rules adding the deduced subformulas to $\Gamma$
      Execute applicable CUR rules adding the deduced subformulas to $\Gamma$
      Execute applicable SIMPLIF rules adding the deduced subformulas to $\Gamma$
    EndWhile
    If $() \in \Gamma$ then return("UNSAT") Else return("SAT")
```

Theorem 4.4.2 $CORSAT(SATALG_P) \wedge PSAT(SATALG_P)$.

Proof. It is trivial that the previous algorithm is sound because it executes only rules that are in the original Logical Calculus. Now let us prove $LCEQUSAT(LC, SATALG_P)$. Concerning the completeness of the algorithm, one can check that the difference with respect to the Logical Calculus is that in the algorithm the inference rules are executed in a certain order. Aiming at proving $LCEQUSAT(LC, SATALG_P)$, we need to prove that the order followed by the algorithm preserves the completeness of the Logical Calculus. To this end, we need to prove that if we execute first a certain inference rule the applicable inference rules before such execution remain applicable after the execution. For instance, assume that we execute first a DUR rule. Thus, if we execute first a DUR rule with (p) and $\mathcal{F}$ the resulting subformula $\mathcal{F}_p$ is added to Γ and no subformula is eliminated from Γ. Thus, inference rules applicable before the execution of the DUR rule remain applicable after executing it. The same reasoning can be done when the first inference executed is CUR, SIMPLIF or AE. Therefore, this proves $LCEQUSAT(LC, SATALG_P)$ and this, together with $SRC(LC)$ and corollary 1, leads to $CORSAT(SATALG_P)$. Now, we shall prove $PSAT(SATALG_P)$.

1. It could be checked that the maximal number of executable DUR and CUR inference rules is bounded by the number n of proposition occurrences in the original formula because each time that a DUR or a CUR is applied, a negative occurrence of a proposition or a whole conjunction is removed.

2. As the size of the inferred sub-formula $\mathcal{F}_i$ is bounded by the size of the original formula $F_0 = \Gamma$ in an inference rule, we have:

$$< F_0, F_1, \ldots, F_n >, \text{ with } F_0 = \Gamma, F_{i+1} = F_i \wedge \mathcal{F}_i, F_n = F_{n-1} \wedge \Box,$$

where:

$$size(F_0) = n$$

$$size(F_1) = size(F_0) + size(\mathcal{F}_0) < size(F_0) + size(F_0) = 2.n$$

$$size(F_i) = size(F_0) + size(\mathcal{F}_0) + size(\mathcal{F}_1) + \ldots + size(\mathcal{F}_{i-1}) <$$

$$size(F_0) + size(F_0) + size(F_0) + \ldots + size(F_0).$$

$$size(F_i) < (i+1).n$$

$$size(F_n) \in O(n^2)$$
$$\forall i, 1 \leq i \leq n, size(F_i) \in O(n^2)$$

In complexity terms, the length of an inferred formula created by $SATALG_P$ increases in $O(n^2)$.

3. On the other hand, as the inferred formula increases in $O(n^2)$, the search time for the two involved clauses in an inference rule requires to scan the current inferred formula and hence, it is in $O(n^2)$.
4. Having searched the two involved clauses, executing with them an inference step can be done in a time bounded by $O(n)$, the time required to copy a sub-formula $\mathcal{F}_i$.
5. Then, searching and executing one inference rule is limited by $O(n^2)$.
6. As the number of inferences is bounded by $size(F_0) = n$, the total complexity of the construction of an **Refutation proof** in $SATALG_P$ is in $O(n^3)$.
7. Therefore, that proves that the SIMPLE-HORN-NNF SAT problem is in $\mathcal{P}$.

■

Altogether, a straight computer implementation in a SAT algorithm $SATALG_P$, of the generation of **Refutation Proofs** corresponding to the defined Logical Calculus, is in $O(n^3)$. This proves the MAIN THEOREM 2, i.e. that the SIMPLE-HORN-NNF SAT problem is polynomial.

4.4.4 A quadratic SIMPLE-HORN-NNF SAT algorithm

In this section, we are going to prove our MAIN THEOREM 3, namely

MAIN THEOREM 3: $\exists x, CORSAT(x) \wedge QSAT(x)$

The first step aiming at an optimal complexity procedure is to avoid the copies of sub-formulas. Thus, we are going to rewrite the Logical Calculus in an "algorithmic way" by writing each inference rule in a way such that, each application of an inference rule will reduce the size of the formula.

As can be checked observing the inference rules, the inferred formula (the consequent part) is formed from the original one (the antecedent part) by eliminating some sub-formulas, more specifically literals or conjunctions, in the original formula. This fact allows us to redefine the inference rules by making explicit which sub-formulas should be eliminated.

So, in this case, the complexity of searching for a pair of clauses becomes in $O(n)$ and $O(n)$ is also, the maximal time required to explore the formula in order to remove a literal $\neg p$, a conjunction C^- or a term CNF^-. Therefore, as the maximal number of inferences is bounded by n, the process complexity is improved and it becomes in $O(n^2)$ in the worst case.

Quadratic Algorithmic Version of the Logical Calculus (QAVLC)

Let us take the nucleus of our Logical Calculus, and rewrite them according to the criterion of algorithmic efficiency.

Definition 4.4.15 *Let us $\neg p$ (resp. C_j^-) be a disjunct in D_j^- (resp. DNF^-) which in turn belongs to sub-formula Γ_i. We note $RemoveLiteral(\neg p, D_j^-, \Gamma_i)$, (resp. $RemoveConjunction(C^-, DNF, \Gamma_i)$) the algorithmic function that* **removes physically** *the occurrence of the literal $\neg p$ in D^- (resp. the whole conjunction C^- in the DNF^- term) from Γ_i by acting directly on the data structure representing the sub-formula Γ_i. Similarly, $RemoveCNFterm(\Box, CNF^-, \Gamma_i)$* **removes physically** *a CNF^- term of Γ_i if $\Box \in CNF^-$.*

Definition 4.4.16 *The new inference rules using functions Remove are called $RCNF$, RL, and RC for respectively $RemoveCNF$ ($SIMPLIF$), $RemoveLiteral$ (DUR) and $RemoveConjunction$ (CUR), and they are defined as follows:*

$$\frac{\Gamma_i = (CNF^- = \{D_1^- \wedge \ldots \wedge \Box \wedge \ldots \wedge D_M^-\} \vee DNF^- \vee C^+)}{\Gamma_i = RemoveCNF(CNF^-, \Gamma_i)}(RCNF)$$

$$\frac{(p_j),\Gamma_i{=}(\{D_1^-\wedge...\wedge D_k^-{=}(\neg p_1\vee...\vee\neg p_j\vee...\neg p_N)\wedge...\wedge D_M^-\}\vee DNF^-\vee C^+)}{\Gamma_i{=}RemoveLiteral(\neg p_j,D_k^-,\Gamma_i)}\ \textit{(RL)}$$

$$\frac{(p_j),\Gamma_i{=}(CNF^-\vee DNF^-{=}[C_1^-\vee...\vee C_k^-{=}\{\neg p1\wedge...\wedge\neg p_j\wedge...\wedge\neg p_N\}\vee...C_M^-]\vee C^+)}{\Gamma_i{=}RemoveConjunction(C_k^-,DNF^-,\Gamma_i)}\ \textit{(RC)}$$

Corollary 4.4.1 *Let us $QAVLC = \{RCNF, RL, RC, AE\}$.*

$$SRC(QAVLC)$$

Proof. It is trivial that:

$$\forall I, I \in LC, \exists J, J \in QAVLC, I(\Gamma) \equiv J(\Gamma)$$

$$\forall J, J \in QAVLC, \exists I, I \in LC, J(\Gamma) \equiv I(\Gamma)$$

As initially the input formula for LC and $QAVLC$ is the same Γ, by induction of the previous two statements on n, the number of Formulas (the length) of a Refutation Proof, we have

$$\forall \Gamma' \exists \Gamma'', \Gamma \vdash_{LC} \Gamma', \Gamma \vdash_{QAVLC} \Gamma'' \text{ and } \Gamma' \equiv \Gamma''$$

$$SRC(LC) \wedge \text{ previous assertions } \Longrightarrow SRC(QAVLC)$$

■

Another step forward before obtaining an Algorithmic Version of the Logical Calculus consists in applying DUR with (p_1), as many times as occurrences of $\neg p_1$ exist in disjunctions D_i^- integrated in a CNF term $\{D_1 \wedge D_2^- \wedge \ldots \wedge D_k^-\}$ of any sub-formula Γ_i. Afterward, the CUR follows with the same principle: removing all the conjunctions C^- containing $\neg p$ from a sub-formula Γ_i.

Definition 4.4.17 *Let us* $\neg p$ *be a disjunct in* D^- *(resp. conjunct in* C^-*) which in turn is in a* CNF^- *(resp.* DNF^-*) term. We note* $RemAllLitSubForm(\neg p, \Gamma_i)$ *(resp.* $RemAllConjSubForm(\neg p, \Gamma_i)$*) the algorithmic function that* **removes physically** *all the occurrences of the literal* $\neg p$ *(resp. conjunctions containing a* $\neg p$ *occurrence) from* Γ_i*, by acting directly on the data structure representing the sub-formula* Γ_i*.*

This new function leads to new inference rules.

Definition 4.4.18 *The new inference rules using functions* $RemAllLitSubForm$ *and* $RemAllConjSubForm$ *are called* $RALSF$ *and* $RACSF$ *and are defined as follows:*

$$\frac{(p), \Gamma_i = (CNF^- \vee DNF^- \vee C^+)}{\Gamma_i = RemAllLitSubForm(\neg p, \Gamma_i)}(RALSF)$$

$$\frac{(p), \Gamma_i = (CNF^- \vee DNF^- \vee C^+)}{\Gamma_i = RemAllConjSubForm(\neg p, \Gamma_i)}(RACSF)$$

Corollary 4.4.2 $RALSF$ *is sound.*

Proof. The proof of the soundness of $RALSF$ follows from the soundness of RL and the definition of the algorithmic function $RemoveAllLitSubForm$.

$RALSF$ is strictly a sequence of RL rules:

$$S = (RL_1, RL_2, \dots, RL_n)$$

such that

RALSF removes $\neg p_i \in D_i$
iff
$\exists RL \in S$ that removes $\neg p_i \in D_i^-$.

As each RL_i is sound, by the corollary 4.4.1,
RALSF is hence sound.

■

Corollary 4.4.3 $RACSF$ *is sound.*

Proof. The proof is analogous to the proof of $RALSF$. ■

Definition 4.4.19 Quadratic Algorithmic Version of the LC (QAVLC) *We redefine the Quadratic Algorithmic Version of the Logical Calculus as the set of inferences rules formed by:* $QAVLC = \{RCNF, RALSF, RACSF, AE\}$.

Theorem 4.4.3 Correctness $\forall \Gamma, \Gamma \vdash_{QAVLC} \Box \iff UNSAT(\Gamma)$

Lemma 4.4.7 Soundness $\forall \Gamma, \Gamma \vdash_{QAVLC} \Box \Longrightarrow UNSAT(\Gamma)$.

Proof. RCNF and AE are the same rules as previously defined in 4.4.16 and 4.4.12 respectively. On the other hand, the soundness of $RALSF$ and $RACSF$ have been established in the last two corollaries. ■

Lemma 4.4.8 Refutation Completeness $UNSAT(\Gamma) \Rightarrow \Gamma \vdash_{QAVLC} \Box$

Proof. The Completeness of the current Logical Calculus (QAVLC) derives straightforwardly from the Completeness of the previous calculus. Indeed, there are only two differences with respect to the calculus as defined in definition 4.4.12.

1. **New clauses are not generated now.** Instead, a formula Γ_i in Γ is transformed in a sub-formula f_i, i.e. Γ is substituted by

   $$\Gamma' = \Gamma * \{\Gamma_i / f_i\}$$

   where $\Gamma * \{\Gamma_i / f_i\}$ denotes the formula resulting of substituting in Γ its subformula Γ_i by the formula f_i.

   Each transformation causes to change the original formula Γ by a new one Γ' by applying one of the four inference rules. As the AE is the same for the LC and for $QAVLC$, we have to prove that the SAT equivalence is preserved in the transformation caused by the other three algorithmic rules $RCNF$, $RALSF$ and $RACSF$:

   (a) By definition,

   $$\frac{\Gamma_i = (CNF^- = \{\Box \wedge D_2^- \wedge \ldots D_k^-\} \vee DNF^- \vee C^+)}{\Gamma_i \leftarrow (DNF^- \vee C^+)}(RCNF)$$

   Hence, it is trivial that $\Gamma_i \equiv RCNF(\Gamma_i)$

   (b) By definition,

   $$\frac{(p), \Gamma_i = (CNF^- \vee DNF^- \vee C^+)}{\Gamma_i * \{CNF^- \leftarrow \mathit{RemoveAllLiterals}(\neg p, CNF^-)\}}(RALSF)$$

   Therefore, it is trivial that $(p) \wedge \Gamma_i \equiv RALSF(\neg p, \Gamma_i)$

   (c) finally, by definition,

   $$\frac{(p), \Gamma_i = (CNF^- \vee DNF^- \vee C^+)}{\Gamma_i * \{DNF^- \leftarrow \mathit{RemoveAllConjunctions}(\neg p, DNF^-)\}}(RACSF)$$

   Thus, it is straightforward that $(p) \wedge \Gamma_i \equiv RACSF(\neg p, \Gamma_i)$

(d) Thus, if Γ is modified in Γ' by one of the inferences I in $LC = \{AE, SIMPLIF, DUR, CUR\}$ and Γ is modified in Γ'' by the corresponding inference J in $QAVLC = \{AE, RACNF, RALSF, RACSF\}$, then $\Gamma' \equiv \Gamma''$. In addition, as the logical equivalence relationship $\equiv$ implies the SAT equivalence, Γ' is SAT equivalent to Γ''.

The previous statement indicates that there is no need for copies of subformulas. Thus, the formula in an automated proof is composed by a set of unit clauses $(p_1), (p_2), \ldots, (p_n)$ and a formula Γ' obtained from the original formula Γ and according to the logical equivalences shown in the three previous points, we have

$$(p_1) \wedge (p_2) \wedge \ldots \wedge (p_n) \wedge \Gamma' \equiv \Gamma$$

2. The order of the applications of the rules has been restricted. Now a particular rule is applied several times, each time with a different occurrence of the same negated literal, i.e. the negated literal of the proposition in a unit clause. But the order considered is the same that the order in the previous definition of the algorithm $SATALG_P$ and as it was proved such order did not prevent completeness (it acts only improving efficiency, see proof of theorem 4.4.2).

■

The definitive algorithmic inferences derived from the Logical Calculus and which will serve to design an efficient SAT algorithm are the following.

Definition 4.4.20 *We note $RemoveAllLiterals(\neg p, \Gamma)$ (resp. $RemoveAllConjunctions(\neg p, \Gamma)$) the algorithmic function that* **removes physically** *all the occurrences of the literal $\neg p$ (resp. conjunctions containing a $\neg p$ occurrence) from the formula Γ, by acting directly on the data structure representing Γ. The algorithmic function $RemoveAllCNFs$* **removes physically** *all the CNF terms containing the empty clause $\Box$.*

These new functions lead to new inference rules.

Definition 4.4.21 *The new inferences rule are called RAL, RAC and $RACNF$ for respectively $RemoveAllLiterals$, $RemoveAllConjunctions$ and $RemoveAllCNF$ and are defined as follows:*

$$\frac{(p), \Gamma}{\Gamma = RemoveAllLiterals(\neg p, \Gamma)} (RAL)$$

$$\frac{(p), \Gamma}{\Gamma = RemoveAllConjunctions(\neg p, \Gamma)} (RAC)$$

$$\frac{\Gamma}{RemoveAllCNFs(\Box, \Gamma)} (RACNF)$$

Theorem 4.4.4 Correctness. *Let us QAVLC={RAL, RAC, RACNF, AE}.*

$$\forall \Gamma, \Gamma \vdash_{QAVLC} \Box \Longleftrightarrow UNSAT(\Gamma)$$

Proof. The proof is straightforward from the previous correctness of theorem 4.4.3. ■

A Correct Quadratic Algorithm

Once established the Quadratic Algorithmic Version of the Logical Calculus, we can design a first algorithm which is a strict materialization of the mechanization of the inference rules in the $QAVLC$ (last definition).

When all the sub-formulas Γ_i of Γ contain a negative sub-formula NNF^- then Γ is trivially satisfiable: a model is obtained by assigning 0 to all the propositional variables. So assume that some positive conjunctions C^+ are present in Γ. Thus, applying the AE rule over these conjunctions C^+ in Γ, produces unit clauses (p). Altogether, this leads to the statement that: the formula Γ is satisfiable or otherwise, some unit clauses can be deduced.

Thus, the next step is to apply the inference rules defined above with the unit clauses. This process is repeated until either no more unit clauses are generated, or the empty clause is produced. In the first case, the formula is satisfiable and in the second one, it is unsatisfiable.

The principle of the algorithm is the following. First, the AE inference is applied to positive conjunctions C^+ and the propositions in the deduced unit clauses are pushed in a stack (function $ApplyAE(\Gamma, Stack)$). For each proposition in the $Stack$, the rules RAL, RAC and $RACNF$ are applied throughout the respective algorithmic functions *RemoveAllLiterals*, *RemoveAllConjunctions* and *RemoveAllCNFs*. If as a consequence of the *Remove* applications, some subformulas become positive conjunctions C^+, the AE rule is applied upon C^+ adding new propositions to the *Stack*. This process finishes when the *Stack* becomes empty, which means that there is no more applicable inferences. Thus, if in such situation, the empty clause $\Box$ has not been deduced, the input formula Γ is satisfiable and else it is unsatisfiable.

```
$SATALG_{Q1}(\Gamma)$
1 ApplyAE($\Gamma$, Stack)
2 While $Stack \neq \{\}$ do:
3     $p \leftarrow pop(Stack)$
4     RemoveAllLiterals($\neg p, \Gamma$)
5     RemoveAllCNFs($\Box, \Gamma$)
6     RemoveAllConjunctions($\neg p, \Gamma$)
7     ApplyAE($\Gamma$, Stack)
8 EndWhile
9 If $() = \mathcal{F} \in \Gamma$ then return("UNSAT") Else return("SAT")
```

Theorem 4.4.5 Correctness *$SATALG_{Q1}(\Gamma)$ returns "UNSAT" iff Γ is unsatisfiable.*

Proof. This theorem is a direct consequence of the Soundness and Refutation Completeness of the QAVLC, stated in theorem 4.4.4.

Indeed, we have the following parallelism. The line 1 starts the executions of inferences with AE. The lines 4,5, 6 and 7 correspond respectively to the applications of the RAL, $RACNF$, RAC and AE rules. The order of application of the inferences does not prevent completeness (see theorem 4.4.2).

When the line 9 is executed, the "while" iteration is finished. But the condition for termination of the "while" loop is verified when there is no more inference applicable. So, as indicated in line 9, Γ is satisfiable iff the empty clause has not been deduced. So we have $LCALGSAT(QAVLC, SATALG_{Q1})$. The proof continues as follows:

1. $\forall x, y, LCALGSAT(x, y) \wedge SRC(x) \Longrightarrow CORSAT(y)$, corollary 3.3.1
2. $LCALGSAT(QAVLC, SATALG_{Q1})$, by the construction of $SATALG_{Q1}$,
3. $SRC(QAVLC)$, proved in 4.4.4.

We obtain: $CORSAT(SATALG_{Q1})$. ■

Theorem 4.4.6 *The complexity of the algorithm $SATALG_{Q1}$ is $O(n^2)$.*

Proof. The execution of each line 4, 5, 6 and 7 takes a time bounded by $O(n)$. As the number maximum of iterations is also bounded by $\Sigma_{C^+ \in \Gamma} |C^+| < O(n)$, therefore the complexity is in $O(n^2)$. ■

Thus, we have proved our

MAIN THEOREM 3: $\exists x, CORSAT(x) \wedge QSAT(x)$

where x is the previously described algorithm $SATALG_{Q1}$.

Our next goal is to design a suitable data structure for the QAVLC and prove that a strictly linear algorithm can be derived from it.

4.4.5 A Linear SIMPLE-HORN-NNF-SAT Algorithm

The goal of this section is to prove the last main theorem:

MAIN THEOREM 4: $\exists x, CORSAT(x) \wedge LSAT(x)$

Before describing the final linear algorithm, we describe first a preliminary almost linear **incorrect** algorithm containing all the data structure, except two subtle data structure to be presented later. This is done as a previous step in order to help to understand the final correct linear SAT algorithm. So, firstly we describe the required data structure and then, we give the corresponding algorithm.

Data Structure. To each proposition p_k, we associate two lists of pointers $D^-(k)$ and $C^-(k)$. Each element (i,j) in $D^-(k)$ (resp. $C^-(k)$) is a pointer to a disjunction D_j^- (resp. conjunction C_j^-) in Γ_i. For each sub-formula Γ_i, we note $C^+(i)$ the list of positive propositions in the C^+ term of Γ_i.

Input: $\Gamma = \Gamma_1 \wedge \Gamma_2 \wedge \ldots \wedge \Gamma_n$
Output: "SAT" or "UNSAT".

Initialization procedure. In this step, all the data structure, i.e. $D^-(k)$, $C^-(k)$ and $C^+(i)$, are initialized according to their definition and the input formula Γ.

```
$SATALG_{Q2}(\Gamma)$
1 $\forall \Gamma_i = (C^+)$ do: $\forall p \in C^+$ do: $push(p, Stack)$
2 While $Stack \neq \{\}$ do:
3     $p_k \leftarrow pop(Stack)$
4     $\forall (i,j) \in D^-(k)$ do: $Remove(\neg p_k, D_{i,j})$
5     $\forall (i,j) \in D^-(k)$ do: If $D_{i,j} = \emptyset$ then $Remove(CNF_i^-, \Gamma_i)$
6     $\forall (i,j) \in C^-(k)$ do: $Remove(C_{i,j}, \Gamma_i)$
7     $\forall (i,j) \in D^-(k) \bigcup C^-(k)$ do: If $\Gamma_i = (C_i^+)$: $\forall p \in C^+(i)$ $push(p, Stack)$
8 Endwhile
9 If $\exists i, 1 \leq i \leq n$ with $\Gamma_i = ()$ then return("UNSAT") Else return("SAT")
```

Theorem 4.4.7 *$SATALG_{Q2}(\Gamma)$ returns "SAT" iff Γ is satisfiable.*

Proof. The parallelism between the instructions of this algorithm and those of the previous one is straightforward. For example, previous line:

$$4 \text{ RemoveAllLiterals}(\neg p, \Gamma).$$

is substituted by the current line:

$$4\ \forall (i,j) \in D^-(p) \text{ do: } Remove(p, D_{i,j}).$$

Both lines have the same effects: removing occurrences of $\neg p$ from disjunctions in the input formula Γ. In the former this statement is clearly verified. In the latter, one can check that the occurrences of $\neg p$ are exactly those indicated by list $D^-(p)$, and line 4 removes exactly these occurrences. The same analysis can be done to prove the algorithmic equivalence of lines 5, 6, and 7 of the previous algorithm and the current one which lead to the proof of the theorem as follows:

1. $\forall x, y, SATEQU(x, y) \wedge CORSAT(x) \Longrightarrow CORSAT(y)$, corollary 4.3.2
2. $SATEQU(SATALG_{Q1}$,$SATALG_{Q2})$, by the construction of both algorithms
3. $CORSAT(SATALG_{Q2})$, by previous theorem 4.4.7

and hence, we obtain $CORSAT(SATALG_{Q2})$.

■

Theorem 4.4.8 *$SATALG_{Q2}$ is in $O(n^2)$*

Proof. It is based on the two following statements:

- Each pair (i,j) in $D^-(k)$ or $C^-(k)$ is associated with an occurrence of a proposition variable in the formula Γ. Hence, the number of iterations is less than n, the size of Γ.
- Each *Remove* function can be done in $O(n)$ time because to remove a sub-formula we need only to scan the formula.

■

The complexity can be improved by executing the *Remove* functions in $O(1)$ time. This is achieved by introducing two counters in the data structure.

- **RAL: removing $\neg p$ occurrences.** To each disjunction D_j^- in clause Γ_i, a counter $Counter(i, j)$ is associated. A decrement of $Counter(i, j)$ indicates the removal of a falsified literal $\neg p$ in D_j^-.
- **RACNF: removing CNF^- terms.** If any $Counter(i, j)$ is set to 0 means that the disjunction D_j^- in Γ_i is falsified and hence the whole CNF^- term is also falsified. This is implemented by setting to 0 a flag $CNF(i)$ associated with each subformula Γ_i.
- **RAC: removing conjunctions C^-.** A counter $Counter.DNF(i)$ is associated with the DNF^- term in each clause Γ_i. Each decrement of $Counter.DNF(i)$ represents a removal of a falsified conjunction C^- in the DNF^- term.

Remark Notice that we do not need to know exactly which literal occurrences (resp. conjunctions) have been removed. What we need to know is merely how many elements have been removed in order to detect when a disjunction (resp. a whole CNF^- term) has been removed totally, i.e. has been falsified, to subsequently apply the And-Elimination rule to the new positive clause C^+.

With these new data structures, the corresponding algorithm is:
Input: $\Gamma = \Gamma_1 \wedge \Gamma_2 \wedge \ldots \wedge \Gamma_n$
Output: "SAT" or "UNSAT".

Initialization procedure. In addition to the previous data structure ($D^-(k)$, $C^-(k)$ and $C^+(i)$), $Counter(i,j)$ and $Counter.DNF(i)$ are initialized according to their definition and the input formula Γ.

```
$SATALG_L(\Gamma)$
1  $\forall\Gamma_i = (C^+)$ do: $\forall p \in C^+$ do: $push(p, Stack)$
2  While $Stack \neq \{\}$ do:
3      $p_k \leftarrow pop(Stack)$
4      $\forall(i,j) \in D^-(k)$ do: Decrement $Counter(i,j)$
5      $\forall(i,j) \in D^-(k)$ do: If $Counter(i,j) = 0$ then $CNF^-(i) \leftarrow 0$
6      $\forall(i,j) \in C^-(k)$ do: Decrement $Counter.DNF^-(i)$
7      $\forall(i,j) \in (D^-(k) \bigcup C^-(k))$ and $CNF^-(i) = Counter.DNF(i) = 0$ do:
           $\forall p \in C^+(i)$ do: $push(p, Stack)$
8  Endwhile
9  If $\exists i, 1 \leq i \leq n$, with $CNF(i) = Counter.DNF(i) = 0$ and $C^+(i) = ()$ then:
       return("UNSAT")
11 Else return("SAT")
```

The structure of the last algorithm is the same that the previous one. The sequence of applications of the inference rules is strictly the same in both algorithms. The parallelism between the operations performed by this algorithm and those of the previous Algorithmic Structure is straightforward.

However, the last algorithm is not correct. Indeed, when we apply the inference by removing elements, there were not exist the problem of modifying erroneously the data structure representing the deduced formula. However, with the counters, that can happen because as the application of the inferences is done via the decrement of counters, for a same unit clause (p) derived several times, a counter can be decremented several times. Actually, we have two types of errors in the previous algorithm:

- Given that a same proposition p can be present in several C^+ terms of the clauses Γ_i, each proposition p can be introduced in the stack several times. To avoid to execute several times the inference rules with the same proposition, we employ a flag $First(p)$. Thus, if $First(p) = 1$ indicates that the inference rules with clause (p) have not been applied and then, these are allowed. Once the inference rules with clause (p) are applied, the flag $First(p)$ is set to 0 disallowing further executions of the inference rules with the same unit clause.

- The second problem comes from the decrements of $Counter.DNF(i)$. Each decrement must correspond to the removal (i.e. falsification) of one conjunct of the DNF^- term in Γ_i. This counter should be set to 0 only when all the conjuncts are falsified. Nevertheless, in the previous algorithm, the deductions of n literals that could belong to a same conjunct C^- of the DNF_i^- term, implies n decrements of $Counter.DNF(i)$. Thus, if the number of conjunctions is n, the counter is set to 0, indicating that the DNF^- term has been removed, and however not all the conjuncts in the DNF^- have been falsified. To overcome this problem, we use a flag call $First(C^-)$ for each conjunct C^- in the DNF^-. Initially, this flag is set to 1. After the first falsification of any literal in C^-, the flag is set to

0. In this way, only one decrement of $Counter.DNF$ is allowed for each conjunct C^- in DNF^-.

Thus, correcting the previous algorithm with the previous defined data structure, we have:

```
If $First(i,j)$= 1 then do: decrement $Counter.DNF(i)$
                            $First(i,j) \leftarrow 0$
```

The resulting algorithm is therefore:

```
$SATALG_L(\Gamma)$
1  $\forall(C^+) \in \Gamma$ do: $\forall p \in (C^+)$ do: $push(p, Stack)$
2  While $Stack \neq \{\}$ do:
3      $p_k \leftarrow pop(Stack)$
4      If $First(k) = 1$ then do:
5          $First(k) \leftarrow 0$
6          $\forall(i,j) \in D^-(k)$ do: Decrement $Counter(i,j)$
7          $\forall(i,j) \in D^-(k)$ do: If $Counter(i,j) = 0$ then $CNF^-(j) \leftarrow 0$
8          $\forall(i,j) \in C^-(k)$ do: If $First(i,j) = 1$ then do:
9              Decrement $Counter.DNF^-(i)$; $First(i,j) \leftarrow 0$
10         $\forall(i,j) \in (D^-(k) \bigcup C^-(k))$ and $CNF(i) = Counter.DNF(i) = 0$ do:
11             $\forall p \in C^+(i)$ do: $push(p, Stack)$
12 Endwhile
13 If $\exists i, 1 \leq i \leq n$, with $CNF(i) = Counter.DNF(i) = 0$ and $C^+(i) = ()$ then
14     return("UNSAT")
15 Else return("SAT")
```

Now, we can ensure the correctness of the algorithm:

Theorem 4.4.9 Correctness. *$SATALG_L(\Gamma)$ returns "SAT" iff Γ is satisfiable.*

Proof. The proof is a consequence of the correctness of the previous $SATALG_{Q2}$ (theorem 4.4.7) and the evident parallelism between the previous instructions and the current ones. Indeed, previous lines 4, 5, 6 and 7, corresponding to inferences RAL, RACNF, RAC and AE respectively, are substituted now by lines 6 (RAL), 7 (RACNF), then 8, 9 and 10 (RAC) and finally 11 (AE).

Then, we have:

1. $\forall x, y, SATEQU(x, y) \wedge CORSAT(x) \Longrightarrow CORSAT(y)$, corollary 4.3.2
2. $SATEQU(SATALG_{Q2}, SATALG_L)$, by the construction of both algorithms
3. $CORSAT(SATALG_{Q2})$, by previous theorem 4.4.7

and hence, we obtain $CORSAT(SATALG_L)$. ■

Concerning the algorithms' complexity, the last algorithm is strictly linear.

Theorem 4.4.10 Complexity *The algorithm $SATALG_L$ is in $O(n)$.*

Proof. It follows from the fact that the maximal number of the main iteration is bounded by n, the number of different proposition variables of the formula and the cost of each iteration is limited by the number of occurrences of a particular proposition variable. Altogether the number of operations executed is bounded by the number of occurrences in the input formula. ■

Following the methodology, we have proved the last theorem:

MAIN THEOREM 4: $\exists x, CORSAT(x) \wedge LSAT(x)$

where x is the previous linear algorithm $SATALG_L$

4.5 Linearity of the HORN-NNF-SAT problem

4.5.1 The HORN-NNF-SAT problem

In this section, we introduce the syntax and semantic elements to define the HORN-NNF-SAT problem. As this problem is a generalization of the previous SIMPLE-HORN-NNF-SAT problem, most of the definitions and concepts given below are straightforward extensions of the previous ones defined in the section corresponding to the linearity of the SIMPLE-HORN-NNF-SAT problem.

We keep the definitions and notations of propositions p, literals L and that of the empty clause "()", or $\Box$, and empty formula $\{\}$. All the other notions and concepts used in the SIMPLE-HORN-NNF-SAT problem require to be adapted to the more general context of the HORN-NNF formulas.

Definition 4.5.1 *A NNF formula is a classical non-clausal formula formed by the well-known induction over the set of propositions* $\mathbf{P}$ *and the connectors* $\neg$*,* $\vee$*, and* $\wedge$ *and with the only restriction that the scope of the negation operator* $\neg$ *is a single proposition. A negative NNF formula, noted* NNF^-*, is a NNF formula formed exclusively by negative literals.*

Example 4.5.1 *The formula below is a NNF formula:*

$$F = \{(p_1) \wedge (p_2) \wedge (p_3) \wedge (p_6)$$
$$\wedge$$
$$(\{(\neg p_1 \vee \neg p_8) \wedge (\neg p_2)\} \quad \vee \quad \{(\neg p_4 \vee \neg p_5) \wedge (\neg p_6)\} \quad \vee \quad \{\neg p_3 \wedge \neg p_7\} \quad \vee \quad \{p_7 \wedge p_5\})$$
$$\wedge$$
$$(\{(\{(\neg p_1 \vee \neg p_8) \wedge (\neg p_2) \wedge (\neg p_6)\} \vee \{\neg p_7\} \vee \{\neg p_1 \wedge \neg p_9\}) \wedge (\neg p_4)\} \vee \{p_8 \wedge p_9\})$$
$$\wedge$$
$$(\neg p_8)\}$$

We can see that the clause in second row is not a SIMPLE-HORN-NNF clause because it has two CNF terms. The clause in the third row is not either a SIMPLE-HORN-NNF formula because it presents a double recursive SIMPLE-HORN-NNF nesting. Indeed it can be written recursively as:

$$((F \vee G \vee H) \wedge (\neg p_4)\} \vee \{p_8 \wedge p_9\})$$

which is a SIMPLE-HORN-NNF formula without DNF^- term, where F, G and H are in turn the respective CNF^- formulas:

$$F = \{(\neg p_1 \vee \neg p_8) \wedge (\neg p_2) \wedge (\neg p_6)\}$$
$$G = \{\neg p_7\}$$
$$H = \{\neg p_1 \wedge \neg p_9\}$$

Thus, the fifth formula has two levels of nested SIMPLE-HORN-NNF formulas.

The graphical representation of the fifth clause of this formula is given in figure 3 following a straightforward extension of the previous principle in section 4.4.1. The conjunct (resp. disjunct) sub-formulas of a conjunction (resp. disjunction) are represented vertically (resp. horizontally).

The set of literals in a simple classical Horn clause included in a HORN-NNF clause can be obtained, similarly as previously, by following a specific path throughout its bi-dimensional representation.

Definition 4.5.2 *A path crosses only one (resp. all the) conjunct (resp. disjunct) sub-formula(s) of a conjunctive (resp. disjunctive) subformula. The only path throughout a literal is the proper literal.*

The indicated path in dashed line correspond to the Horn clause:

$$(\neg p_2 \vee \neg p_7 \vee \neg p_9 \vee p_8)$$

The formula can be also represented graphically by a tree as indicated in figure 5.

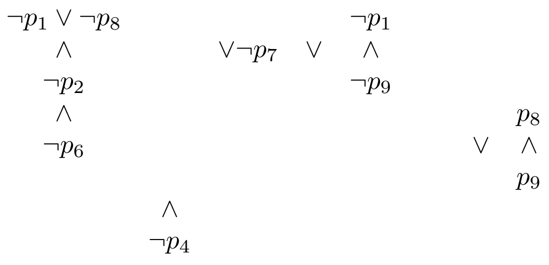


Figure 4.3: Bidimensional representation of the subformula

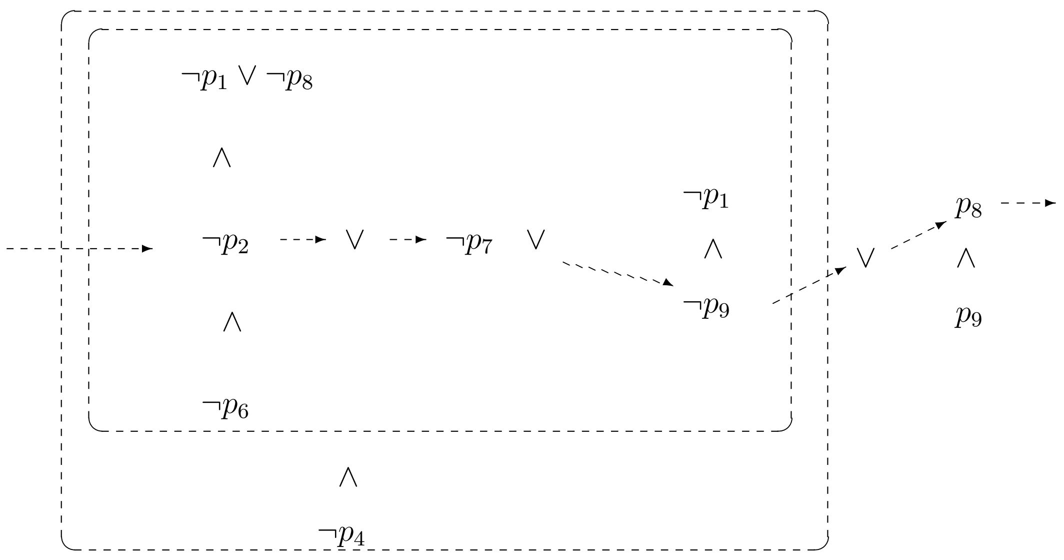


Figure 4.4: A path in the formula

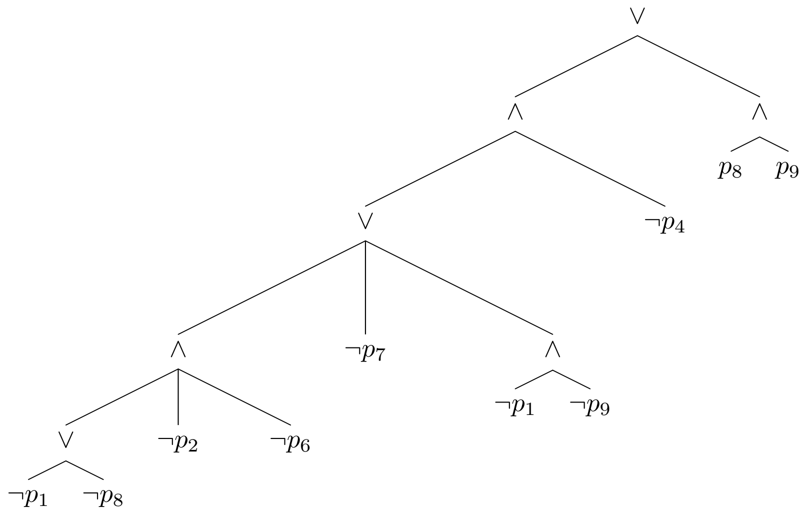


Figure 4.5: The tree of the sixth clause

Definition 4.5.3 *A conjunctive NNF^- formula*

$$\{NNF_1^- \wedge NNF_2^- \wedge \ldots \wedge NNF_k^-\},$$

is said to be in $CNNF^-$ form. Similarly, a disjunctive NNF^- formula:

$$(NNF_1^- \vee NNF_2^- \vee \ldots \vee NNF_k^-),$$

is said to be in $DNNF^-$ form.

Definition 4.5.4 *As an extension of the CNF^- case, any disjunctive $DNNF^-$ formula can be called a NNF^- clause.*

Definition 4.5.5 *A HORN-NNF clause is a clause formed by three optional terms $\mathcal{F} = CNNF^- \vee DNNF^- \vee C^+$. A HORN-NNF formula is a set of HORN-NNF clauses.*

Now, we define the classical semantic concepts related to satisfiability of formulas

Definition 4.5.6 *An interpretation I assigns to each formula Γ one value in the set $\{0, 1\}$ and its satisfiability property is extended from SIMPLE-HORN-NNF formulas to HORN-NNF formulas as follows. An interpretation I satisfies:*

- *A NNF^- conjunctive formula $CNNF^- = \{NNF_1^- \wedge \ldots \wedge NNF_k^-\}$, iff $I(NNF_i^-) = 1$ for every NNF_i^-.*
- *A NNF^- disjunctive formula $DNNF^- = (NNF_1^- \vee \ldots \vee NNF_k^-)$, iff $I(NNF_i^-) = 1$ for at least one NNF_i^-.*

- *A HORN-NNF clause $\mathcal{F} = (CNNF^- \vee DNNF^- \vee C^+)$ iff $I(CNNF^-) = 1$ or $I(DNNF^-) = 1$ or $I(C^+) = 1$.*
- *A HORN-NNF formula iff I satisfies all its HORN-NNF clauses.*

An interpretation I is a model *of a formula Γ if it satisfies the formula. We say that Γ is* satisfiable *if it has at least one model, otherwise, it is* unsatisfiable.

Definition 4.5.7 *By definition, the formula $\Gamma = \{()\}$ and the clause $() \equiv \Box$ are unsatisfiable; the formula $\{\}$ is satisfiable.*

Definition 4.5.8 *The HORN-NNF-SAT problem is the problem of deciding whether a HORN-NNF formula is satisfiable.*

Definition 4.5.9 *A clause $\mathcal{F}$ is a logical consequence of a formula Γ, noted $\Gamma \models \mathcal{F}$, iff every model of Γ is a model of $\mathcal{F}$.*

4.5.2 Logical Calculus

The Logical Calculus for the HORN-NNF formulas is obtained by substituting terms D^-, C^-, CNF^- and DNF^- in the previous four rules, by general NNF^- formulas. Also, the level of nesting allowed can be of any finite order.

As in the simple case, we need two kinds of Unit Resolution rules:

- DUR, for the case where the negated literal $\neg p$ of a unit clause (p) is in a disjunction, and
- CUR, similarly when $\neg p$ is in a conjunction.

In order to extend the Logical Calculus of the SIMPLE-HORN-NNF-SAT problem to the HORN-NNF-SAT problem, we take separately each inference rule and we generalize the terms D^-, C^-, CNF^- and DNF^- existing in the inference rule by replacing them appropriately by general NNF^- terms.

Definition 4.5.10 *The rule DUR for the HORN-NNF formulas where the old terms have been substituted by general NNF^- formulas is the following:*

$$\frac{(p), (\{(\neg p \vee NNF_1^-) \wedge NNF_2^-\} \vee NNF_3^- \vee C^+)}{(\{(NNF_1^-) \wedge NNF_2^-\} \vee NNF_3^- \vee C^+)}(DUR)$$

Now we need to extend the nesting level from 3 to any order k:

$$\frac{(p), (\{(\ldots(\{(\neg p \vee NNF_1^-) \wedge NNF_2^-\} \vee NNF_3^-) \wedge \ldots) \wedge NNF_{k-1}^-\} \vee NNF_k^- \vee C^+)}{(\{(\ldots(\{(NNF_1^-) \wedge NNF_2^-\} \vee NNF_3^-) \wedge \ldots) \wedge NNF_{k-1}^-\} \vee NNF_k^- \vee C^+)}(DUR)$$

Remark: The term NNF_k can be a conjunct or a disjunct depending only on the number k of layers of nesting. If k is even, NNF_k is a conjunct and else, if k is odd, it is a disjunct.

Thus, the new inference rule antecedent are obtained from the previous ones by:

- substituting in the rule antecedent, the terms $(\neg p_2 \vee \ldots \vee \neg p_n)$, $\{D_2^- \wedge \ldots \wedge D_k^-\}$ and DNF^- by general NNF^- formulas, and by
- generalizing from the level 3 of nesting of the operators $\vee/\wedge$ in formulas SIMPLE-HORN-NNF to a general level k of nesting in HORN-NNF.

Then, with the antecedent defined, the consequent is easily derived. Applying the previous principle to the three inference rules of the previous LC for the SIMPLE-HORN-NNF formulas, we obtain the definitive inference rules for the general HORN-NNF case.

Definition 4.5.11 *The Logical Calculus for the general HORN-NNF formulas is the following:*

$$\frac{(\{(\ldots\{(\{\Box \wedge NNF_1^-\} \vee NNF_2^-) \wedge NNF_3^-\} \vee \ldots \vee NNF_{k-1}^-) \wedge NNF_k^-\} \vee C^+)}{(\{(\ldots\{(NNF_2^-) \wedge NNF_3^-\} \vee \ldots \vee NNF_{k-1}^-) \wedge NNF_k^-\} \vee C^+)}(SIMPLIF)$$

$$\frac{(p), (\{(\ldots(\{(\neg p \vee NNF_1^-) \wedge NNF_2^-\} \vee NNF_3^-) \wedge \ldots) \wedge NNF_{k-1}^-\} \vee NNF_k^- \vee C^+)}{(\{(\ldots(\{(NNF_1^-) \wedge NNF_2^-\} \vee NNF_3^-) \wedge \ldots) \wedge NNF_{k-1}^-\} \vee NNF_k^- \vee C^+)}(DUR)$$

$$\frac{(p), (\{(\ldots\{(\{\neg p \wedge NNF_1^-\} \vee NNF_2^-) \wedge NNF_3^-\} \vee \ldots) \wedge NNF_{k-1}^-\} \vee NNF_k^- \vee C^+)}{(\{(\ldots\{(NNF_2^-) \wedge NNF_3^-\} \vee \ldots) \wedge NNF_{k-1}^-\} \vee NNF_k^- \vee C^+)}(CUR)$$

$$\frac{(\{p_1 \wedge \ldots \wedge p_i \wedge \ldots \wedge p_n\})}{(p_1) \wedge \ldots \wedge (p_i) \wedge \ldots \wedge (p_n)}(AE)$$

Example 4.5.2 *Let us take the previous example:*

$$F = \{(p_1) \wedge (p_2) \wedge (p_3) \wedge (p_6)$$

$$\wedge$$

$$(\{(\neg p_1 \vee \neg p_8) \wedge (\neg p_2)\} \quad \vee \quad \{(\neg p_4 \vee \neg p_5) \wedge (\neg p_6)\} \quad \vee \quad \{\neg p_3 \wedge \neg p_7\} \quad \vee \quad \{p_7 \wedge p_5\})$$

$$\wedge$$

$$(\{(\{(\neg p_1 \vee \neg p_8) \wedge (\neg p_2) \wedge (\neg p_6)\} \vee \{\neg p_7\} \vee \{\neg p_1 \wedge \neg p_9\}) \wedge (\neg p_4)\} \vee \{p_8 \wedge p_9\})$$

$$\wedge$$

$$(\neg p_8)\}$$

$$\frac{(p2), (\{(\neg p_1 \vee \neg p_8) \wedge (\neg p_2)\} \vee \quad \{(\neg p_4 \vee \neg p_5) \wedge (\neg p_6)\} \vee \{\neg p_3 \wedge \neg p_7\}) \vee \{p_7 \wedge p_5\})}{(\{(\neg p_1 \vee \neg p_8) \wedge \Box\} \vee \quad \{(\neg p_4 \vee \neg p_5) \wedge (\neg p_6)\} \vee \{\neg p_3 \wedge \neg p_7\} \vee \{p_7 \wedge p_5\})}(DUR)$$

$$\frac{(\{(\neg p_1 \vee \neg p_8) \wedge \Box\} \vee \quad \{(\neg p_4 \vee \neg p_5) \wedge (\neg p_6)\} \vee \{\neg p_3 \wedge \neg p_7\} \vee \{p_7 \wedge p_5\})}{\{(\neg p_4 \vee \neg p_5) \wedge (\neg p_6)\} \vee \{\neg p_3 \wedge \neg p_7\} \vee \{p_7 \wedge p_5\})}(SIMPLIF)$$

$$\frac{(p_6)(\{(\neg p_4 \vee \neg p_5) \wedge (\neg p_6)\} \vee \{\neg p_3 \wedge \neg p_7\} \vee \{p_7 \wedge p_5\})}{\{(\neg p_4 \vee \neg p_5) \wedge \Box\} \vee \{\neg p_3 \wedge \neg p_7\} \vee \{p_7 \wedge p_5\})}(DUR)$$

$$\frac{\{(\neg p_4 \vee \neg p_5) \wedge \Box\} \vee \{\neg p_3 \wedge \neg p_7\} \vee \{p_7 \wedge p_5\})}{\{\neg p_3 \wedge \neg p_7\}) \vee \{p_7 \wedge p_5\})}(SIMPLIF)$$

$$\frac{(p3), (\{\neg p_3 \wedge \neg p_7\} \vee \{p_7 \wedge p_5\})}{(\{p_7 \wedge p_5\})}(CUR)$$

$$\frac{(\{p_7 \wedge p_5\})}{(p_7), (p_5)}(AE)$$

$$\frac{(p_1), (\{(\{(\neg p_1 \vee \neg p_8) \wedge (\neg p_2) \wedge (\neg p_6)\} \vee \{\neg p_7\} \vee \{\neg p_1 \wedge \neg p_9\}) \wedge (\neg p_4)\} \vee \{p_8 \wedge p_9\})}{(\{(\{(\neg p_8) \wedge (\neg p_2) \wedge (\neg p_6)\} \vee \{\neg p_7\} \vee \{\neg p_1 \vee \neg p_9\}) \wedge (\neg p_4)\} \vee \{p_8 \wedge p_9\})}(DUR)$$

$$\frac{(p_1), (\{(\{(\neg p_8) \wedge (\neg p_2) \wedge (\neg p_6)\} \vee \{\neg p_7\} \vee \{\neg p_1 \wedge \neg p_9\}) \wedge (\neg p_4)\} \vee \{p_8 \wedge p_9\})}{(\{(\{(\neg p_8) \wedge (\neg p_2) \wedge (\neg p_6)\} \vee \{\neg p_7\}) \wedge (\neg p_4)\} \vee \{p_8 \wedge p_9\})}(CUR)$$

$$\frac{(p_2), (\{(\{(\neg p_8) \wedge (\neg p_2) \wedge (\neg p_6)\} \vee \{\neg p_7\}) \wedge (\neg p_4)\} \vee \{p_8 \wedge p_9\})}{(\{(\{(\neg p_8) \wedge \Box \wedge (\neg p_6)\} \vee \{\neg p_7\}) \wedge (\neg p_4)\} \vee \{p_8 \wedge p_9\})}(DUR)$$

$$\frac{(\{(\{(\neg p_8) \wedge \Box \wedge (\neg p_6)\} \vee \{\neg p_7\}) \wedge (\neg p_4)\} \vee \{p_8 \wedge p_9\})}{(\{(\{\neg p_7\}) \wedge (\neg p_4)\} \vee \{p_8 \wedge p_9\})}(SIMPLIF)$$

$$\frac{(p_7), \{(\{\neg p_7\}) \wedge (\neg p_4)\} \vee \{p_8 \wedge p_9\})}{(\{\Box \wedge (\neg p_4)\} \vee \{p_8 \wedge p_9\})}(DUR)$$

$$\frac{(\{\Box \wedge (\neg p_4)\} \vee \{p_8 \wedge p_9\})}{(\{p_8 \wedge p_9\})}(SIMPLIF)$$

$$\frac{(\{p_8 \wedge p_9\})}{(p_8), (p_9)}(AE)$$

$$\frac{(\neg p_8), (p_8)}{\Box}(DUR)$$

Now, we can state the first main theorem.

Theorem 4.5.1 *The previous Logical Calculus is Sound and Refutation Complete for the HORN-NNF formulas.*

Proof. The soundness is trivial and the completeness theorem proof is very similar to that of the logical calculus for the SIMPLE-HORN-NNF-SAT problem. Let us take as an example the first lemma equivalent to lemma 4.4.3.

Lemma 4.5.1 *If* $\Gamma^+ \models p$ *and* $rank(p) = k$ *then there exists at least one clause* $\mathcal{F} = (CNNF_1^- \vee CNNF_2^- \vee \ldots \vee CNNF_k^- \vee \{DNNF_1^- \wedge DNNF_2^- \wedge \ldots \wedge DNNF_n^-\} \vee \mathcal{C}^+)$ *such that:*

1. $p \in \mathcal{C}^+$*, or similarly,* $\mathcal{F} \in pos(p)$
2. $\forall \neg p_i \in \mathcal{F}$*,* $rank(p_i) < k$
3. $\forall CNNF_i^-$*,* $\exists DNNF^-$*,* $DNNF^- \in CNNF_i^-$ *s.t.* $\Gamma^+ \models \neg DNNF^-$ *and* $\exists DNNF_j^-$*,* $\forall CNNF^-$*,* $CNNF^- \in DNNF_j^-$ *s.t.* $\Gamma^+ \models \neg CNNF^-$*.*

Proof.

1. Assume $\nexists \mathcal{F}, \mathcal{F} \in pos(p)$. Let us I be a model of Γ^+ such that $I(p) = 1$. Let us consider an interpretation I' with the same mapping that I except for $I'(p) = 0$. Obviously I is also a model of Γ^+ because I' satisfies more literals than I since all the occurrences of p in Γ^+ are negated. Hence, $\Gamma^+ \not\models p$.

2. Assume that $\exists \neg p_i$ such that $rank(p_i) = l \geq k$. Then by definition of rank, $rank(p) = l + 1 > k$.

3. Assume that $\exists CNNF_i$, $\forall DNNF^-$, $DNNF^- \in CNNF_i^-$ such that $\Gamma^+ \not\models \neg DNNF^-$ or assume that $\forall DNNF_j$, $\exists CNNF^-$, $CNNF^- \in DNNF_j^-$, $\Gamma^+ \not\models \neg CNNF^-$. We provide the proof for the first alternative, the proof for the second one is completely similar. If $\neg DNNF^-$ is not a logical consequence means that there is a model I of Γ^+ such that $I(DNNF^-) = 1$ and then $I(CNNF_i^-) = 1$. But, taking into account this fact, defining I' equal to I except for the mapping $I'(p) = 0$ is also a model and then we have $\Gamma^+ \not\models p$.

■

The remaining lemmas of the proof are proved similarly using the same kind of generalization followed in the previous lemma with respect to the equivalent lemma in the MAIN THEOREM 1 for the SIMPLE-HORN-NNF-SAT problem.

■

Thus, **MAIN THEOREM 1:** $\exists x, SRC(x)$ is true for the previous LC, namely we have, $SRC(LC)$.

4.5.3 A Quadratic Algorithm

The algorithmic version of the current Logical Calculus is the same that the previous one, defined in 4.4.20.

$$\frac{(p), \Gamma}{\Gamma = RemoveAllLiterals(\neg p, \Gamma)} (RAL)$$

$$\frac{(p), \Gamma}{\Gamma = RemoveAllConjunctions(\neg p, \Gamma)} (RAC)$$

$$\frac{\Gamma}{\Gamma = RemoveAllCNFs(\Box, \Gamma)} (RACNF)$$

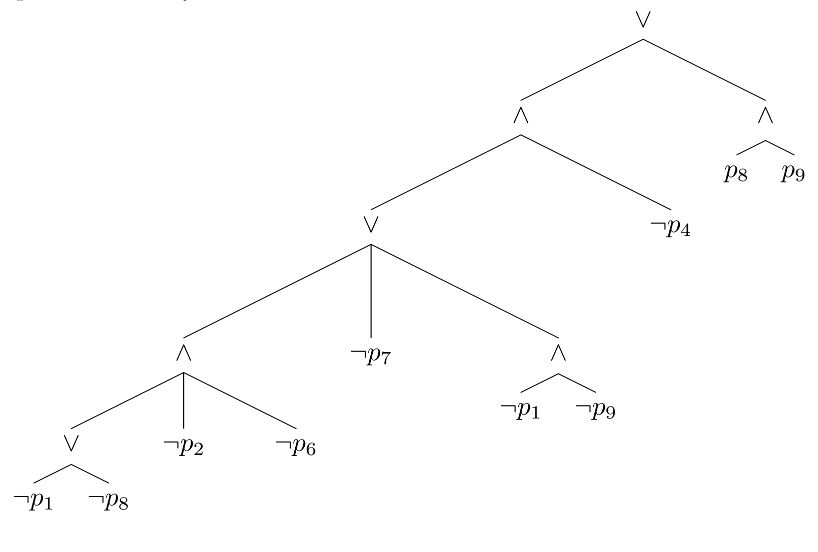
As the calculus is the same that the previous one, the corresponding quadratic algorithm differs in only one point with respect to the previous one for the SIMPLE-HORN-NNF-SAT problem.

In the current HORN-NNF-SAT problem the nesting of formulas oblige to propagate the situations appearing when disjunctions are converted in the empty clause $\Box$. Removing terms in disjunctions can make reduce a disjunction to the empty clause. To its turn, this empty clause can be a term in a conjunction that contains the removed disjunction, and removing this conjunction could yield an empty disjunction. Thus, propagation of the empty clause in the formulas is required.

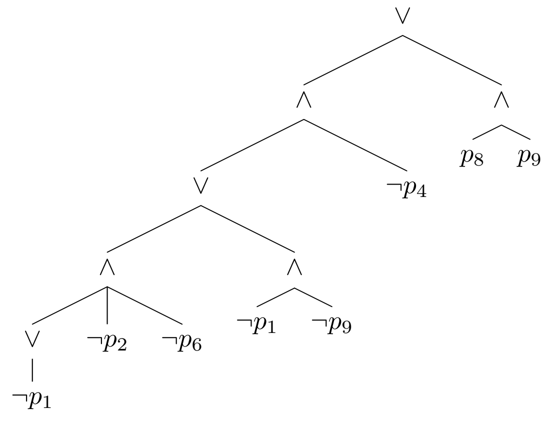
Let us take an example. Consider the sixth subformula of the previous HORN-NNF formula:

$$(\{(\{(\neg p_1 \vee \neg p_8) \wedge (\neg p_2) \wedge (\neg p_6)\} \vee \{\neg p_7\} \vee \{\neg p_1 \wedge \neg p_9\}) \wedge (\neg p_4)\} \vee \{p_8 \wedge p_9\})$$

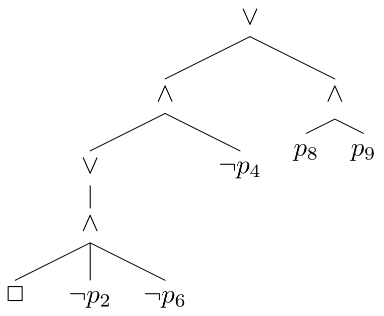
Its representation by a tree is :



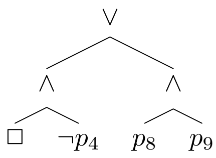
Assume that firstly unit clauses (p_8), (p_7) have been deduced. After applying the correspondent inferences, the resulting formula becomes:



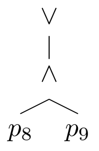
If now $(p1)$ is deduced, the formula is simplified as follows:



Now propagating the clause $\square$, we obtain the tree:



Now propagating again the empty clause $\square$ leads to the tree:



These last steps of propagation of the $\square$ make the difference between the previous algorithms and the current ones for the HORN-NNF-SAT problem. The following algorithm is a quadratic algorithm for the HORN-NNF-SAT problem:

$SATALG_{Q1}(\Gamma)$

```
1  ApplyAE(Γ, Stack)
2  While $Stack \neq \{\}$ do:
3    $p \leftarrow pop(Stack)$
4    RemoveAllLiterals($\neg p, \Gamma$)
5    RemoveAllCNFs($\Box, \Gamma$)
6    RemoveAllConjunctions($\neg p, \Gamma$)
7    Propagate($\Box$, Γ)
8    ApplyAE(Γ, Stack)
9  EndWhile
10 If $() = \mathcal{F} \in \Gamma$ then return("UNSAT") Else return("SAT")
```

Theorem 4.5.2 *The previous algorithm is correct.*

Proof. The correctness of the algorithm follows from that of $SATALG_{Q1}$ for the SIMPLE-HORN-NNF SAT problem. ■

Theorem 4.5.3 *The previous algorithm is quadratic.*

Proof. The quadratic complexity is derived from the quadratic complexity of the algorithm $SATALG_{Q1}$ for the SIMPLE-HORN-NNF-SAT problem. Indeed the function "Propagate" is linear like the other operations in lines 4, 5, and 6, and as the maximal number of iterations is limited by the number of different propositions, the global complexity is quadratic. ■

This proves the MAIN THEOREM 3 for the HORN-NNF-SAT problem:

$$CORSAT(SATALG_{Q1}) \wedge QSAT(SATALG_{Q1})$$

4.5.4 A Linear Algorithm for the HORN-NNF-SAT problem

Before defining the data structure, we introduce some notations needed.

If a subformula α in Γ is of the form $\alpha = \alpha_1 \vee \alpha_2 \vee \ldots \vee \alpha_i \vee \ldots \vee \alpha_k$ or $\alpha = \alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_i \wedge \ldots \wedge \alpha_k$ then we will consider that α contains α_i or, similarly, α_i is included in α. Also, from a practical point of view, we will consider that α is the father of each α_i and that the set $\{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ are the successors of α. Each subformula α is implemented by associating a connector, namely a set of pointers, to the set of successors and to each successor α_i a pointer toward its father. This father will be noted $D^-(\alpha_i)$ if α_i is a disjunctive term in α or $C^-(\alpha_i)$ if α_i is a conjunctive term in α.

Input: $\Gamma = \Gamma_1 \wedge \Gamma_2 \wedge \ldots \wedge \Gamma_n$
Output: "SAT" or "UNSAT".

Initialization procedure. In this procedure, all the data structure are initialized, namely $D^-(k)$ and $C^-(k)$ are pointers to the occurrences of proposition p_k appearing in disjunctions and conjunctions respectively. Now, in addition, there are also pointers $D^-(\alpha_i)$ or $C^-(\alpha_i)$ of each subformula α_i to the father formula α that contains it. Conversely, there is a list of pointers $Succ(\alpha)$ associated to each formula α toward the successor formulas α_i.(Like previously, $[\alpha]$ denotes a pointer to the formula α).

The following is a SAT algorithm with quadratic complexity that employs the above described data structure.

```
$SATALG_{Q2}(\Gamma)$
1  $\forall\Gamma_i = (C^+)$ do: $\forall p \in C^+$ do:  $push(p, Stack)$
2  While $Stack \neq \{\}$ do:
3      $p_k \leftarrow pop(Stack)$
4      $\forall[\alpha] \in D^-(k)$ do: $Remove(\neg p_k, \alpha)$
5      $\forall[\alpha] \in D^-(k)$ do: If $\alpha = \emptyset$ then PROPAGATE($[\alpha]$)
6      $\forall[\alpha] \in C^-(k)$ do: $Remove(\alpha, D^-(\alpha))$
7      $\forall[\alpha] \in C^-(k)$ do: If $Succ(D^-(\alpha)) = \emptyset \vee Succ(D^-(\alpha)) = (C^+)$ then
                                   PROPAGATE($[D^-(\alpha)]$)
8  Endwhile
9  If $\exists i, 1 \leq i \leq n$ with $\Gamma_i = ()$ then return("UNSAT") Else return("SAT")
10 End
```

```
**PROPAGATE**($[\beta]$){$\beta$ is a $\vee$ node}
1  If $Succ(\beta) = (C^+)$ then do: $\forall p \in C^+$ do:  $push(p, Stack)$
2  Else If $C^-(\beta) \neq \Gamma$ do:
3      $\gamma \leftarrow C^-(\beta)$; $Remove(\gamma, D^-(\gamma))$
4      If $Succ(D^-(\gamma)) = \emptyset \vee Succ(D^-(\gamma)) = (C^+)$ then PROPAGATE($[D^-(\gamma)]$)
5  End
```

Theorem 4.5.4 *$SATALG_{Q2}$ together with only line 1 of PROPAGATE is correct for the SIMPLE-HORN-NNF SAT problem.*

Proof. The proof derives straightforwardly from the correctness of the first algorithm in section 4.4.5 for the SIMPLE-HORN-NNF-SAT problem and the parallelism of the above algorithm and the mentioned algorithm in section 4.4.5. Indeed, the only difference is reduced to the different notations employed in both algorithms.

■

Theorem 4.5.5 *$SATALG_{Q2}$ and $PROPAGATE$ form a correct decision SAT algorithm for the HORN-NNF SAT problem.*

Proof. Due to the construction of $SATALG_{Q1}$ and $SATALG_{Q2}$ algorithms, it is easy to check that $SATEQU(SATALG_{Q1}, SATALG_{Q2})$. Then, we have:

1. $\forall x, y, SATEQU(x, y) \wedge CORSAT(x) \Longrightarrow CORSAT(y)$, corollary 4.3.2
2. $SATEQU(SATALG_{Q1}$,$SATALG_{Q2})$, by the construction of both algorithms

3. $CORSAT(SATALG_{Q1})$, by previous theorem 4.5.2

and hence, we obtain $CORSAT(SATALG_{Q2})$ ■

Theorem 4.5.6 *$SATALG_{Q2}$ and $PROPAGATE$ decide in $O(n^2)$ whether a formula is Satisfiable.*

Proof. Firstly, we analyze $SATALG_{Q2}$, then $PROPAGATE$ and finally the global complexity. Thus, the complexity of "remove" operations in the $SATALG_{Q2}$ function is in $O(n)$, where n is the size of Γ. On the other hand, the maximal number of iterations of this function is bounded by the total number of propositional variable occurrences in the C^+ terms of sub-formulas Γ_i. Thus, $SATALG_{Q2}$ is in $O(n^2)$ in the worst-case. Concerning the $PROPAGATE$ procedure, each recursion is associated to an arc in a tree representing a subformula. Once the arc has been crossed, it is removed with the "remove" function. This warrants that each arc representing a subformula is crossed only once. Thus, $PROPAGATE$ is in $O(k)$, where k is the total number of subformulas in Γ. Altogether, $SATALG_{Q2}(\Gamma)$ ends in $O(n^2)$ time. ■

Having obtained a quadratic algorithm, the next step is to optimize it with appropriate data structure in order to obtain a strictly linear complexity algorithm. The data structures are chosen similarly like in the case SIMPLE-HORN-NNF to advance from the $SATALG_{Q2}$ to $SATALG_L$. More concretely we employ counters for processing removal in disjunctions and flags to prevent repeating inferences.

Data Structures $First(k)$ is a flag associated to each proposition p_k. It avoids to repeat the same inference with a proposition p_k more than once. A flag $First(\alpha)$ is associated with each conjunctive subformula $\alpha = \alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n$. Its use is the same that for the SIMPLE-HORN-NNF-SAT problem. $First(D^-(\alpha))$ prevent removing more than one conjunction from the disjunction $D^-(\alpha)$ when several deduction propagations belonging to the same conjunction are carried out. Finally, there is a counter associated with each disjunctive subformula $\alpha = \alpha_1 \vee \alpha_2 \vee \ldots \vee \alpha_n$, $Counter(\alpha)$. Its role, like in the disjunctions in the SIMPLE-HORN-NNF case, is to keep the number of disjunct terms not removed yet by effect of the different deductions.

```
$SATALG_L(\Gamma)$
1  $\forall \Gamma_i = (C^+)$ do: $\forall p \in C^+$ do: $push(p, Stack)$
2  While $Stack \neq \{\}$ do:
3      $p_k \leftarrow pop(Stack)$
4      If $First(k) = 1$ then do:
5          $First(k) \leftarrow 0$
6          $\forall [\alpha] \in D^-(k)$ do: Decrement $Counter(\alpha)$
7          $\forall [\alpha] \in D^-(k)$ do: If $Counter(\alpha) = 0$ then PROPAGATE($[\alpha]$)
8          $\forall [\alpha] \in C^-(k)$ do: If $First(\alpha) = 1$ then do:
9              $First(\alpha) \leftarrow 0$; Decrement $Counter(D^-(\alpha))$
10         $\forall [\alpha] \in C^-(k)$ do: If $Counter(D^-(\alpha)) = 0$ then PROPAGATE($D^-(\alpha)$)
```

```
11 Endwhile
12 If $\exists \Gamma_i, 1 \leq i \leq n$, with $Counter(\Gamma_i) = 0$ then return("UNSAT") Else return("SAT")
13 End
```

PROPAGATE($[\beta]$)$\{\beta$ is a disjunctive formula$\}$

```
1  If Succ($\beta$) = ($C^+$) then do: $\forall p \in C^+$ do: $push(p, Stack)$
2  Else If $C^-(\beta) \neq \Gamma$ do:
3      If $First(C^-(\beta)) = 1$ then do:
4            $First(C^-(\beta))) \leftarrow 0$
5            Decrement $Counter(D^-(C^-(\beta)))$
6            If $Counter(D^-(C^-(\beta))) = 0$ then PROPAGATE($D^-(C^-(\beta))$)
7  End
```

Theorem 4.5.7 *$SATALG_L(\Gamma)$ returns "SAT" iff Γ is satisfiable.*

Proof. This function is like the previous one with the following substitutions: 1) Remove functions have been transformed by counter decrements and, 2) Tests for empty formulas have been transformed by tests for 0 in the corresponding counters associated to the corresponding formulas. Hence, by construction of $SATALG_L$, $SATEQU(SATALG_{Q2}, SATALG_L)$ is obtained. Then, we have:

1. $\forall x, y, SATEQU(x, y) \wedge CORSAT(x) \Longrightarrow CORSAT(y)$, corollary 4.3.2
2. $SATEQU(SATALG_{Q2}$,$SATALG_L)$, by the construction of both algorithms
3. $CORSAT(SATALG_{Q2})$, by previous theorem 4.5.5

and hence, we obtain $CORSAT(SATALG_L)$ ■

Theorem 4.5.8 *$SATALG_L$ and $PROPAGATE$ form a SAT decision procedure with strictly $O(n)$ complexity.*

Proof. Each line in the $SATALG_L$ iteration is executed in time proportional to the number of occurrences of a particular proposition p_k in the formula. The number of iterations is bounded by the number m of different propositions in the formula. Hence, the operations executed in the whole set of iterations is in $O(n)$, namely proportional to the number of occurrences of propositional variables in the input formula Γ.

The execution of the first line in the PROPAGATE function is executed as many times as occurrences of propositions exist in the C^+ parts of subformulas Γ_i. The remaining operations of PROPAGATE are executed in constant time. Thus, we have to check that the total number of calls to the PROPAGATE function is at most proportional to the number of symbols in the input formula Γ.

A call to the function PROPAGATE is executed each time the counter associated with a disjunctive formula is set to 0. Then, the PROPAGATE function is executed at most the number of disjunctive subformulas existing in the input formula. Thus, the number of calls to the PROPAGATE function is in $O(k)$, where k is the number of $\vee$ symbols in the input formula Γ.

Thus, the whole algorithm $SATALG_L - PROPAGATE$ is in $O(n + k)$, where n and k are respectively the number of propositional occurrences and the number of $\vee$ symbols in the input formula Γ.

■

The last two theorems prove the MAIN THEOREM 4 for the general HORN-NNF SAT problem:

$$CORSAT(SATALG_L) \wedge LSAT(SATALG_L)$$

Thus, the main result of this chapter is proved.

4.6 Conclusions

On the theoretical side, our contribution described here aims at pushing further the frontiers of non-clausal tractability. Thus, we firstly have defined a new class of formulas in Negation Normal Form having a Horn-like shape. In this sense, the proposed formulas absorb the Horn language as a particular case. Secondly, we have established a set of inference rules which are sound and refutationally complete. In third place, we have designed strictly linear algorithms to solve the satisfiability problem in this class of formulas.

On the practical side, as the formulas keep a Horn-like structure, they are of relevant interest in such applications as for instance those based in Rule Based Systems. Indeed, the rules and the questions of many real applications require to represent and reason with a richer language than the Horn formulas language. The proposed formulas represent logically equivalent pure Horn problems but with exponentially less symbols. Hence, as the described algorithm runs in linear time on this class, the gain of time can be of an exponential order with respect to the known linear algorithms running on the Horn formulas.

Chapter 5

The Regular HORN-NNF SAT problem

5.1 Introduction

Solving the SAT problem in many-valued logics is an important challenge due to the repercussions in many different areas of computer Science such as Theorem Proving, Approximated Reasoning, Hardware Design, Deductive Data Bases, Automated Software Validation, Planning, Logic Programming, Knowledge-based Control Systems, etc. The interest of considering many-valued logic instead of classical logic lies mainly in the fact that many valued logics can cope with certain uncertainty aspects existing almost always in real world applications. For a survey on Many-valued Automated Deduction issues the reader can see [49].

For these reasons, at the beginning of the last decade, Automated Deduction in many-valued logics emerged as a new and promising research topic for the AI community.

In this chapter, we will deal with the non clausal signed logic SAT problem. Signed logic is a kind of many-valued logic that is an extension of the classical logic in the following sense. Atoms in propositional bi-valued logic are noted by p and $\neg p$. Knowing that the set of truth values is $\{0, 1\}$, these atoms could be written also as $\{1\}\!:\!p$ and $\{0\}\!:\!p$ respectively. Thus, in a many-valued case, if N is the set of truth values an atom in signed logic is denoted $S\!:\!p$, where $S \subseteq N$, and its negated proposition is $N/S\!:\!p$.

Regular logic is a particular case of signed logic with two assumptions 1) N is a total ordered set $\{0, \frac{1}{N-1}, \frac{2}{N-1}, \ldots, 1\}$ and, 2) the set S can be either of positive polarity or negative polarity. Positive (resp. negative) polarity means that S takes the interval of values comprised between a given value $j \in N$ and 1 (resp. 0 and the given value j).

The satisfiability relation varies only w.r.t. the literal level. Namely, an interpretation I satisfies $S : p$ iff $I(p) \in S$ and it satisfies a conjunction (resp. disjunction) of formulas iff it satisfies each (resp. at least one) formula of the conjunction (resp. disjunction).

Although the extension from classical to signed logic seems quite straightforward, signed Automated Deduction is recognized to require the use of techniques associated with non classical logics. Thus, the signed SAT problem treated here is considered to be within the Non-classical Automated Deduction field.

The signed SAT problem keeps a particular relevance with respect to any other many-valued SAT problem. This is because in [47] has been proved that a SAT problem expressed in any finite many-valued logic can be transformed into an equivalent signed SAT problem in polynomial time. This means that a solver of the signed SAT problem can act as a general many-valued SAT solver. Indeed, in order to solve a many-valued SAT problem first one could transform the problem into a signed SAT problem and then applying the signed SAT solver. So, any finite many-valued SAT problem can be solved by means of the signed SAT solver. Thus, advances in the signed SAT problem have direct consequences on any finite many-valued SAT problem. In other words, the signed SAT problem could be seen as a representative problem of the many-valued SAT problems.

This chapter is structured as follows. In the next section we present a survey of the related work. In section three, we describe the methodology utilized to prove that the Regular SIMPLE-HORN-NNF-SAT problem can be solved in polynomial time. In section four (resp. five), we apply this methodology to prove the almost linearity (resp. quadratic complexity) of the Regular SIMPLE-HORN-NNF-SAT (resp. HORN-NNF-SAT) problem and we provide a almost linear (resp. quadratic) algorithm to solve it. Finally, we explain the conclusions we have obtained in this research work.

5.2 Related Work

We review successively the main works concerning signed and regular logic, non-clausal tractability and many-valued tractability.

Some relevant works related to signed and regular logic.
Signed logic was formally developed by Hähnle [43] and Lu, Murray and Rosenthal [61] for Automated Deduction with tableau approaches and resolution in the context of finitely-valued logics. The notion of a signed formula was first introduced by Smullyan in [79] within the classical logic context. In [44] a generalization of this concept to the many-valued logic is given. Later [45] defined the concepts of regular signs and regular formulas. Murray and Rosenthal [70] developed Resolution for signed formulas. In [33] the notion of Horn many-valued formula was introduced by the first time and it was proved that the Horn SAT problem for a sub-case of the regular logic is almost linear. In [48], and later in [63], the regular unit resolution rule is given and proved that it is refutationally complete for regular Horn clauses. [50] proves the completeness of regular connected tableaux and NC-resolution for non-clausal formulas. [62] analyzes the

cases of first-order signed formulas, annotated and fuzzy logic. [59] describes an inference rule based on resolution for regular many-valued logics which has interesting properties for automated reasoning. [65] analyzes the transition phase phenomenon in the regular CNF satisfiability problem, [16] shows two transformations between signed and classical CNF logics, [13] defines a collection of mappings that transform many-valued clausal forms into satisfiability equivalent Boolean clausal forms. [64] deals with the signed 2-SAT problem and [11] studies the interface between P and NP in Mono+pPartiallySigned-2SAT and Regular+pSigned-2SAT.

Non-clausal Tractability Tractability has attracted much attention, specially in classical logic. As far as we know, the first published results concerning non-clausal tractability comes from [31, 32, 35] where a strictly linear bottom-up algorithm to test the satisfiability of a subclass of non-clausal formulas is detailed. Such a class embeds the Horn case as a particular case. In [40] a linear top-down algorithm is given for the same non-clausal subclass of formulas.

New results concerning non-clausal tractability are reported in [75] where a method called Restricted Fact Propagation is presented which is a quadratic, incomplete non-clausal inference procedure.

More recently, in [81, 82] a significant advance in non-clausal tractability has been accomplished. The author defines a class of formulas by extending the Horn formulas to the field of non-clausal formulas. Such extension relies on the concept of polarity. In [81], a SLD-resolution variant with the property of being refutationally complete is showed, although its computational complexity is not studied. In [82] a method for propositional and some many-valued non-clausal Horn-like formulas is described and it is stated that the method is sound, incomplete and linear.

However, concerning the last issue, no algorithm is specified, indeed the steps of the method are described as different propagations of some truth values in a sparse tree. Then, although it seems that the number of inferences of the proposed method is linear, it is not proved the resulting complexity (w.r.t. the number of computer instructions) of a linear number of truth value propagations on the employed sparse trees.

In a previous work carried out by the authors [2, 7, 6] the linearity of some sub-classes of formulas of the general classical (bi-valued) NNF formulas is proved.

Many-valued tractability The earliest work on this topic is due to [33] where the SAT problem and other related problems for a sub-class of the regular Horn logic is proved to be almost linear. In [48] the regular Horn problem is proved to be also almost linear. Then, in [34] the 2-SAT problem is analyzed proving that the regular 2-SAT and the special case of the signed 2-SAT in which all the signs are singletons are polynomials. In [16] the regular Horn SAT problem where the truth values form a finite lattice is proved to be polynomial.

In [82] the first many-valued non-clausal SAT problem that can be determined in polynomial time has been defined. More recently, another many-valued non-clausal SAT problem with a polynomial complexity has been identified [5, 9].

The many-valued logic and the non-clausal form studied there are sub-cases respectively of the regular logic and the non-clausal form analyzed in this chapter.

5.3 Proof Methodology

In this section we explain our methodology to prove that the Regular SIMPLE-HORN-NNF-SAT problem can be solved in polynomial time. We used in the previous chapter this methodology to prove that the general HORN-NNF-SAT problem of classical (bi-valued) logic can be solved strictly in linear time. Now we extend the application of our methodology by applying a similar strategy to the more complicated case of propositional many valued logic, specifically to the formulas we name as regular HORN-NNF formulas. Our methodology has as an ultimate goal the design of an algorithm which resolves the related Regular HORN-NNF-SAT problem in polynomial time; another objective is to prove progressively the logical properties of the final and complicated algorithm.

5.3.1 Description of the Proof Methodology

This proof is decomposed in four steps: the **first step** consists in defining a Sound and Refutation Complete Logical Calculus LC. In the **second one**, we define a SAT algorithm A1 based on the previous Logical Calculus which is correct but it needs some refinements to optimization complexity aspects. In the **three step**, we specify a better polynomial SAT algorithm A2. Then, the objective of the final **fourth step** is to write a definitive algorithm A3 whose structural design complexity is carefully designed. So, our final polynomial SAT algorithm, namely A3, is the final step of the complexity optimization process, represented by the sequence $< LC, A1, A2, A3 >$.

The idea behind our approach is, conducting the optimization process verifying that each SAT algorithm in the sequence, is satisfiability equivalent to the previous one in the sequence (The equivalence relationship between LC and $A1$ will be specified later).

We describe these steps with a simple First Order Logic formalism. Variables are written in small letters and constants in capital letters. We use four first order predicates:

- SRC, to assert the Soundness and Refutation Completeness of a Logical Calculus x, i.e. $SRC(x)$ is true iff x is Sound and Refutation Complete.
- $CORSAT$, to assert the CORrectness of a SAT algorithm x, namely $CORSAT(x)$ stands for: x is a SAT algorithm that running on a formula Γ returns "UNSAT" iff Γ is unsatisfiable.
- $SATEQU$, to assert that two SAT algorithms are satisfiability equivalents, namely $SATEQU(x_1, x_2)$ says that, for any formula Γ, if x_1 and x_2 are both SAT algorithms and, if x_1 running on Γ returns "UNSAT", then x_2 running on the same formula Γ returns "UNSAT" too.

- $LCEQUSAT(x, y)$, to assert the equivalence between a Logical Calculus x and a SAT algorithm y in the following sense: $LCEQUSAT(x, y)$ is true iff x derives $\Box$ from Γ when y running on the same formula Γ returns "UNSAT".

More formally, noting:

$$UNSAT(\Gamma) \equiv \Gamma \text{ is unsatisfiable}$$

and $x(\Gamma)$ the value returned by algorithm x running on formula Γ, the previous logical predicates are precisely defined respectively by the following equivalences:

$$\forall x, SRC(x) \Longleftrightarrow \forall\Gamma, \Gamma \vdash_x \Box \Leftrightarrow UNSAT(\Gamma)$$

$$\forall x, CORSAT(x) \Longleftrightarrow \forall\Gamma, x(\Gamma) = \text{“}UNSAT'' \Leftrightarrow UNSAT(\Gamma)$$

$$\forall x, y, SATEQU(x, y) \Longleftrightarrow \forall\Gamma, x(\Gamma) = y(\Gamma)$$

$$\forall x, y, LCEQUSAT(x, y) \Longleftrightarrow \forall\Gamma, \Gamma \vdash_x \Box \Leftrightarrow y(\Gamma) ='' UNSAT''$$

With, this notation, the following corollaries are straightforward:

Corollary 5.3.1

$$\forall x, y, LCEQUSAT(x, y) \wedge SRC(x) \Longrightarrow CORSAT(y)$$

Proof. It follows from the definitions of the predicates in the statements. ■

Corollary 5.3.2

$$\forall x, y, SATEQU(x, y) \wedge CORSAT(x) \Longrightarrow CORSAT(y)$$

Proof. It follows from the definitions of the predicates in the statements. ■

As mentioned before, trying to prove the SAT correction of an algorithm with a complex design structure is not the right way to proceed, because the more complex is an algorithm the more difficult is to analyze it.

Thus, in order to prove $CORSAT(A_{i+1})$ directly and exclusively from its structural design, we prove $CORSAT(A_i) \wedge SATEQU(A_i, A_{i+1})$, that together with the previous Corollary, implies $CORSAT(A_{i+1})$. Thus, transferring this approach throughout the optimization sequence $< LC, A1, A2, A3 >$, we have that the difficulty of proving CORSAT(A3) has been reduced to the simpler proofs SRC(LC), LCEQUSAT(LC,A1), SATEQU(A1,A2) and SATEQU(A2,A3).

- The proof of SRC(LC) is a quite standard proof and it is based on known techniques.

- The remaining three proofs are simple ones because we can choice an algorithm whose structural design is close to its predecessor object in the sequential optimization process. Thus, making a straight parallelism between the instructions of the two algorithms, we can state their SAT equivalence.

Thus, in order to prove the correctness of an almost linear Regular SIMPLE-HORN-NNF-SAT algorithm and a quadratic Regular HORN-NNF-SAT algorithm, we prove successively:

1. The existence of a Sound and Refutation Complete Logical Calculus (LC). This proof will be done via well-known mathematical techniques.
2. The existence of a correct SAT algorithm $SATALG_P$ that is a direct implementation of the previous Logical Calculus LC and whose complexity is polynomial.
3. The existence of a correct SAT algorithm $SATALG_Q$ with quadratic complexity issued from $SATALG_P$ via an optimization step.
4. The existence of a SAT correct algorithm $SATALG_{AL}$ with an almost linear complexity issued from $SATALG_Q$ via an optimization step.

Remark. For the Regular HORN-NNF-SAT problem, we have only specified the first three steps.

5.3.2 Theorems of the Proof methodology

We distinguish four MAIN THEOREMS in our proof methodology, one for each step of the proof of polynomiality of the Regular (SIMPLE-)HORN-NNF SAT problem. They are stated below together with a sketch of their proof. In addition to the previous First Order Predicates, we also use $PSAT(x), QSAT(x)$ and $ALSAT(x)$ which state that x is a SAT algorithm of respectively Polynomial, Quadratic and an Almost Linear complexity.

1. **Logical Calculus (LC)** This step consists in establishing a Logical Calculus appropriated to solve the Regular (SIMPLE)-HORN-NNF-SAT problem. Thus, we must prove its Soundness and Refutation Completeness.

   MAIN THEOREM 1: $\exists x, SRC(x)$

   Proof Sketch. The proof consists in finding a particular Logical Calculus LC and proving its Soundness and Refutation Completeness. As it will be shown, this proof follows standard mathematical techniques. ■

2. **Algorithmic Version of the Logical Calculus** This step consists in defining an Algorithmic Version of the Logical Calculus (AVLC), proving its correctness, w.r.t. the satisfiability test, and its polynomial complexity.

   MAIN THEOREM 2: $\exists x, CORSAT(x) \wedge PSAT(x)$

Proof Sketch. The proof is based on the first corollary and it is divided in two steps:

(a) Firstly, we choose a particular $SATALG_P$ for x, namely a polynomial SAT algorithm such that the proof of $LCEQUSAT(LC, SATALG_P)$ is straightforward. Namely, our proposed $SATALG_P$ is a direct algorithmic materialization of the concrete LC defined in the first theorem. Then, applying corollary 5.3.1, we deduce $CORSAT(SATALG_P)$.

(b) Second, the proof of $PSAT(SATALG_P)$ follows from a very simple complexity analyze of the designed $SATALG_P$.

■

3. **Quadratic Algorithm (QA)** This step consists in modelling the variables of the previous $SATALG_P$ with specific data structures, and in proving that there exists a Quadratic Algorithm $SATALG_Q$ derived from $SATALG_P$.

   MAIN THEOREM 3 $\exists x, CORSAT(x) \wedge QSAT(x)$

   Proof Sketch. It is based on corollary 5.3.2. We show that there exists a quadratic algorithm $SATALG_Q$ very similar to the previous $SATALG_P$. Given such similarity, we prove easily the statement $SATEQU(SATALG_P, SATALG_Q)$. Thus, $CORSAT(SATALG_Q)$ follows from $SATEQU(SATALG_P, SATALG_Q)$, $CORSAT(SATALG_P)$, proved in the previous theorem, and the corollary 5.3.2. $QSAT(SATALG_Q)$ is obtained by an easy complexity analyze. ■

4. **An almost linear algorithm.** Finally, the optimal algorithm is described progressively by choosing appropriately the data structures and the computer instructions, aiming at the design of an almost linear algorithm. The substitutions of the initial data structures and the initial computer instructions by new ones are explained accurately.

   MAIN THEOREM 4 $\exists x, CORSAT(x) \wedge ALSAT(x)$

   Proof Sketch. $SATEQU(SATALG_Q, SATALG_{AL})$ follows from the tight similarity between $SATALG_Q$ and $SATALG_{AL}$. $CORSAT(SATALG_{AL})$ is straightforwardly derived from such SAT equivalency, from $CORSAT(SATALG_Q)$, proved in the previous theorem, and from the corollary 5.3.2. ■

Remark The theorem 4 is obtained for the Regular SIMPLE-HORN-NNF-SAT problem. For the regular HORN-NNF SAT problem only the first three theorems apply. The obtaining of the four theorem for this problem is left as future work.

The advantage of the proposed methodology is that the difficulty of obtaining a proof of the logical properties of the final complex algorithm is simplified, substituting the original proof by the proof of the Soundness and Refutation Completeness of a Logical Calculus, which is done as mentioned by standard formal techniques. The other three SAT equivalence proofs are trivial. In that way, we eliminate many possible sources of error when proving our main results, namely that the Regular SIMPLE-HORN-NNF-SAT (resp. HORN-NNF-SAT) problem is almost linear (resp. quadratic) and hence, by showing that there exists a correct and an almost linear (resp. quadratic) algorithm to solve it.

5.4 The Regular SIMPLE-HORN-NNF-SAT problem

In this section we apply the methodology described in the previous section to prove the polynomiality or almost linearity of the Regular SIMPLE-HORN-NNF SAT problem.

5.4.1 Regular SIMPLE-HORN-NNF formulas

Firstly, we introduce the class of Regular SIMPLE-HORN-NNF formulas ending with the definition of the related satisfiability problem. A more detailed discussion about foundations of these concepts can be found in [8, 4, 33, 46, 48].

Definition 5.4.1 Signed literal. *Let $N = \{i_1, i_2, \ldots, i_n\}$ be a finite set of truth values, S a subset of N ($S \subseteq N$), p a proposition and $\leq$ a total order associated with N. An expression of the form $S\!:\!p$ is a signed literal and S is its sign. Given a signed literal $S\!:\!p$ and a set of truth values N, then $(N \setminus S)\!:\!p$ denotes the complement of $S\!:\!p$.*

Definition 5.4.2 Regular sign. *Let $\uparrow i$ denote the set $\{j \in N \,|\, i \leq j\}$ and $\downarrow i$ the set $\{j \in N \,|\, j \leq i\}$, where N is the set of truth values, $\leq$ is a linear order on N and $i \in N$. If a sign S is equal to either $\downarrow i$ or $\uparrow i$, then it is a regular sign. A signed literal $S\!:\!p$ has positive (resp. negative) polarity if $S = \uparrow i$ (resp. $S = \downarrow i$).*

Definition 5.4.3 Regular formulas. *Let R be a regular sign. A regular literal is a signed literal whose sign is regular. A regular clause $\mathcal{C}$ is a disjunction of regular literals $\mathcal{C} = R_1 : p_1 \vee R_2 : p_2 \vee \ldots \vee R_m : p_m$. A regular Horn clause is a regular clause with at most one regular literal with positive polarity. A regular unit clause is a regular clause containing only one regular literal. We denote the empty regular clause $()$ by $\Box$. A regular Horn formula is a conjunction of regular Horn clauses.*

Definition 5.4.4 *A negative disjunction, noted D^-, is a disjunction of regular literals with negative polarity, namely $D^- = (\downarrow i_1 : p_1 \vee \downarrow i_2 : p_2 \vee \ldots \vee \downarrow i_n : p_n)$. Similarly, a negative conjunction C^- is a conjunction of regular literals with negative polarity, i.e. $C^- = \{\downarrow i_1 : p_1 \wedge \downarrow i_2 : p_2 \wedge \ldots \wedge \downarrow i_n : p_n\}$. Also $C^+ = \{\uparrow$*

$i_1 : p_1 \wedge \uparrow i_2 : p_2 \wedge \ldots \wedge \uparrow i_n : p_n\}$ *stands for a conjunction of regular literals with positive polarity.*

Definition 5.4.5 Regular SIMPLE-HORN NNF clauses. *A negative conjunctive normal form, noted CNF^-, is a conjunction of negative disjunctions, i.e. $CNF^- = D_1^- \wedge D_2^- \wedge \ldots \wedge D_k^-$. A negative disjunctive normal form DNF^- is a disjunction of negative conjunctions, namely $DNF^- = C_1^- \vee C_2^- \vee \ldots \vee C_k^-$. A regular SIMPLE-HORN-NNF clause $\mathcal{C}$ is a disjunction of three optional terms $\mathcal{C} = DNF^- \vee CNF^- \vee C^+$. Clauses without the C^+ term are said negative clauses.*

Example 5.4.1 *From the following regular SIMPLE-HORN-NNF clause:*
$\mathcal{C} = ((\{\downarrow 0.2 : p_1 \wedge \downarrow 0.1 : p_2\} \vee \downarrow 0.15 : p_3) \vee \{(\downarrow 0.25 : p_4 \vee \downarrow 0.4 : p_5) \wedge \downarrow 0.2 : p_6\} \vee \{\uparrow 0.8 : p_7 \wedge \uparrow 0.7 : p_8\})$,

we can easily identify:
$DNF^- = (\{\downarrow 0.2 : p_1 \wedge \downarrow 0.1 : p_2\} \vee \downarrow 0.15 : p_3)$
$CNF^- = \{(\downarrow 0.25 : p_4 \vee \downarrow 0.4 : p_5) \wedge \downarrow 0.2 : p_6\}$
$C^+ = \{\uparrow 0.8 : p_7 \wedge \uparrow 0.7 : p_8\}$

Definition 5.4.6 *A Regular SIMPLE-HORN NNF formula Γ is a finite conjunction of Regular SIMPLE-HORN-NNF clauses $\mathcal{C}$. We denote $\Gamma_\Box$ any Γ-formula containing the empty clause.*

Example 5.4.2 *The following formula is an example of a Regular SIMPLE-HORN-NNF formula:*

$$\Gamma = \{(\uparrow 0.7 : p_1) \wedge (\uparrow 0.6 : p_3) \wedge (\uparrow 0.8 : p_6)$$
$$\wedge$$
$$((\{\downarrow 0.2 : p_1 \wedge \downarrow 0.1 : p_2\} \vee \{\downarrow 0.15 : p_3\}) \vee \{(\downarrow 0.25 : p_4 \vee \downarrow 0.4 : p_5) \wedge (\downarrow 0.2 : p_6)\} \vee$$
$$\{\uparrow 0.8 : p_7 \wedge \uparrow 0.7 : p_8\}),$$
$$\wedge$$
$$(\downarrow 0.1 : p_8)\}$$

The fourth (non-unitary) sub-formula is equivalent to the following eight clauses:

$$(\downarrow 0.2 : p_1 \vee \downarrow 0.15 : p_3 \vee \downarrow 0.25 : p_4 \vee \downarrow 0.4 : p_5 \vee \uparrow 0.8 : p_7)$$
$$(\downarrow 0.2 : p_1 \vee \downarrow 0.15 : p_3 \vee \downarrow 0.25 : p_4 \vee \downarrow 0.4 : p_5 \vee \uparrow 0.7 : p_8)$$
$$(\downarrow 0.2 : p_1 \vee \downarrow 0.15 : p_3 \vee \downarrow 0.2 : p_6 \vee \uparrow 0.8 : p_7)$$
$$(\downarrow 0.2 : p_1 \vee \downarrow 0.15 : p_3 \vee \downarrow 0.2 : p_6 \vee \uparrow 0.7 : p_8)$$
$$(\downarrow 0.1 : p_2 \vee \downarrow 0.15 : p_3 \vee \downarrow 0.25 : p_4 \vee \downarrow 0.4 : p_5 \vee \uparrow 0.8 : p_7)$$
$$(\downarrow 0.1 : p_2 \vee \downarrow 0.15 : p_3 \vee \downarrow 0.25 : p_4 \vee \downarrow 0.4 : p_5 \vee \uparrow 0.7 : p_8)$$
$$(\downarrow 0.1 : p_2 \vee \downarrow 0.15 : p_3 \vee \downarrow 0.2 : p_6 \vee \uparrow 0.8 : p_7)$$
$$(\downarrow 0.1 : p_2 \vee \downarrow 0.15 : p_3 \vee \downarrow 0.2 : p_6 \vee \uparrow 0.7 : p_8)$$

The CNF clause represented by a Regular SIMPLE-HORN-NNF sub-formula can be obtained by an exhaustive enumerating of the *paths* (defined below) in a bi-dimensional representation of the formula. The fourth sub-formula of Γ can be represented graphically in two dimensions as follows:

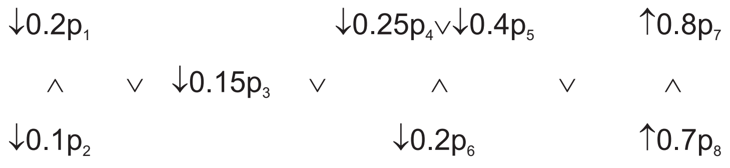


Figure 5.1: Bi-dimensional representation of the fourth clause

Definition 5.4.7 Bi-dimensional Representation *In the graphical representation of a formula, terms in a conjunction (resp. disjunction) are represented vertically (resp. horizontally).*

Definition 5.4.8 *A* formula path *crosses throughout only one (resp. all) literal (resp. literals) that form(s) a conjunction (resp. disjunction).*

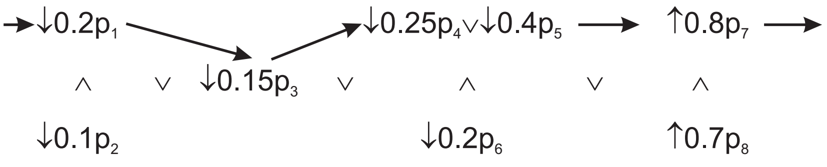


Figure 5.2: A formula path

Example 5.4.3 *In the previous example there are exactly eight* formula paths.

The set of literals in each classical simple clause represented in a factorized way by a Regular SIMPLE-HORN-NNF clause, can be obtained by taking the literals crossed in a specific *formula path* in the graphical representation corresponding to the Regular SIMPLE-HORN-NNF clause.

Example 5.4.4 *The clause corresponding to the path indicated by a continuos line in the previous figure is:*

$$\downarrow 0.2 : p_1 \vee \downarrow 0.15 : p_3 \vee \downarrow 0.25 : p_4 \vee \downarrow 0.4 : p_5 \vee \uparrow 0.8 : p_7$$

Proposition 5.4.1 *There is a bijection between the set of simple clauses represented in a Regular SIMPLE-HORN-NNF clause and the set of paths defined in its bi-dimensional representation.*

The proof is trivial.

Now, we define the classical semantic concepts related to the formula satisfiability problem.

Definition 5.4.9 *An interpretation I assigns to each proposition p one value in the set $N = \{i_1, i_2, \ldots, i_n\}$ and it satisfies:*

- *A regular literal $\downarrow i{:}p$ (resp. $\uparrow i{:}p$) iff $I(p) \leq i$ (resp. $I(p) \geq i$)*
- *A negative disjunction $D^- = (\downarrow i_1{:}p_1 \vee \downarrow i_2{:}p_2 \vee \ldots \vee \downarrow i_n{:}p_n)$, iff $I(p_j) \leq i_j$, for at least one $\downarrow i_j{:}p_j$.*
- *A negative conjunction $C^- = \{\downarrow i_1{:}p_1 \wedge \downarrow i_2{:}p_2 \wedge \ldots \wedge \downarrow i_n{:}p_n\}$, iff $I(p_j) \leq i_j$, for every $\downarrow i_j{:}p_j$.*
- *A positive conjunction $C^+ = \{\uparrow i_1{:}p_1 \wedge \uparrow i_2{:}p_2 \wedge \ldots \wedge \uparrow i_n{:}p_n\}$, iff $I(p_j) \geq i_j$, for every $\uparrow i_j{:}p_j$.*
- *A term $CNF^- = \{D_1^- \wedge \ldots \wedge D_k^-\}$, iff I satisfies D_i^- for every D_i^-.*
- *A term $DNF^- = (C_1^- \vee \ldots \vee C_k^-)$, iff I satisfies at least one C_i^-.*
- *A SIMPLE-HORN-NNF clause $\mathcal{C} = (DNF^- \vee CNF^- \vee C^+)$, iff I satisfies DNF^- or CNF^- or C^+.*
- *A SIMPLE-HORN-NNF formula Γ iff I satisfies all its SIMPLE-HORN-NNF clauses.*

Definition 5.4.10 *An interpretation I is a* model *of a formula Γ if it satisfies the formula. We say that Γ is* satisfiable *if it has at least one model, otherwise, it is* unsatisfiable.

Definition 5.4.11 *By definition, the conjunction $\{()\}$ and the disjunction $() \equiv \Box$ are unsatisfiable. On the other hand, the conjunction $\{\}$ and the disjunction $(\{\})$ are satisfiable.*

Definition 5.4.12 *The Regular SIMPLE-HORN-NNF-SAT problem is the problem of deciding whether a Regular SIMPLE-HORN-NNF formula is satisfiable.*

5.4.2 Logical Calculus

In this sub-section, we are going to prove the first theorem of our proof methodology, namely:

MAIN THEOREM 1: $\exists x, SRC(x)$

The Logical Calculus needed to prove the (un)satisfiability of Regular SIMPLE-HORN-NNF formulas is formed by rules extended from the case of classical SIMPLE-HORN-NNF formulas developed in the previous chapter.

Definition 5.4.13 *The Logical Calculus is formed by four rules. The first one, called $RSIMPLIF$, allows to simplify formulas. The second and third rules are two Positive General Unit Resolution rules, called $RDUR$ and $RCUR$, for respectively Regular Disjunction Unit Resolution and Regular Conjunction Unit Resolution appropriated for the Regular SIMPLE-HORN-NNF formulas. The last rule, called Regular And Elimination rule (RAE), enables to obtain regular unit clauses from the regular positive conjunction (C^+). We recall that both "()" and $\Box$ denote the empty clause.*

$$\frac{(\{\Box \wedge D_2^- \wedge \ldots \wedge D_k^-\} \vee DNF^- \vee C^+)}{(DNF^- \vee C^+)}(RSIMPLIF)$$

$$\frac{i_1 > j_1, (\uparrow i_1 : p_1), (\{(\downarrow j_1 : p_1 \vee \downarrow j_2 : p_2 \ldots \vee \downarrow j_n : p_n) \wedge D_2^- \wedge \ldots \wedge D_k^-\} \vee DNF^- \vee C^+)}{(\{(\downarrow j_2 : p_2 \vee \ldots \vee \downarrow j_n : p_n) \wedge D_2^- \wedge \ldots \wedge D_k^-\} \vee DNF^- \vee C^+)}(RDUR)$$

$$\frac{i_1 > j_1, (\uparrow i_1 : p_1), (CNF^- \vee \{\downarrow j_1 : p_1 \wedge \downarrow j_2 : p_2 \wedge \ldots \downarrow j_n : p_n\} \vee C_2^- \vee \ldots \vee C_k^- \vee C^+)}{(CNF^- \vee C_2^- \vee \ldots \vee C_k^- \vee C^+)}(RCUR)$$

$$\frac{(\{L_1 \wedge \ldots \wedge L_i \wedge \ldots \wedge L_n\})}{(L_1) \wedge \ldots \wedge (L_i) \wedge \ldots (L_n)}(RAE)$$

Remark Note that when we have a unit clause $(\uparrow i : p)$, the computation steps to apply the corresponding inferences depending on the cases that $i > j$ is true and either $(\downarrow j : p)$ is in a regular conjunction C^- or a regular disjunction D^-. Whenever $(\downarrow j : p)$ is in a disjunction D^- (resp. in a conjunction C^- which in turn it is in a disjunction D^-) then, this literal (resp. conjunction C^-) is removed from the disjunction D^- with rule RDUR (resp. with rule RCUR). If this literal (resp. conjunction) was the only one in the original disjunction D^-, or it was the last remaining literal (resp. conjunction) in the initial disjunction D^- because the other original literals (resp. conjunctions) have been removed previously then, the $\Box$ is derived. In other words D^- is transformed in $\Box$ after some removals of regular literals and regular conjunctions in D.

Definition 5.4.14 Refutation Proof
A refutation of a Regular SIMPLE-HORN-NNF formula Γ, is a finite succession of Regular SIMPLE-HORN-NNF formulas

$$< F_1, F_2, \ldots, F_n >, \ \textit{with } F_n = \Box$$

where F_i, $1 \leq i \leq n$, is a subformula in Γ or it is obtained by applying one inference rule of the Logical Calculus in previous definition 5.4.13 upon a unit clause and a sub-formula, both in $F_1, \ldots, F_{i-1}$.

Example 5.4.5 *Let us consider the following formula:*

$$\Gamma = \{(\uparrow 0.7 : p_1) \wedge (\uparrow 0.6 : p_3) \wedge (\uparrow 0.8 : p_6)$$

$$\wedge$$

$((\{\downarrow 0.2 : p_1 \wedge \downarrow 0.1 : p_2\} \vee \{\downarrow 0.15 : p_3\}) \vee \{(\downarrow 0.25 : p_4 \vee \downarrow 0.4 : p_5) \wedge (\downarrow 0.2 : p_6)\} \vee$
$\{\uparrow 0.8 : p_7 \wedge \uparrow 0.7 : p_8\})$

$$\wedge$$

$$(\{(\downarrow 0.3 : p_1 \vee \downarrow 0.3 : p_7) \wedge (\downarrow 0.2 : p_9)\} \vee \{\uparrow 0.2 : p_8 \wedge \uparrow 0.8 : p_9\})$$

$$\wedge$$

$$(\{\downarrow 0.1 : p_8\})\}$$

We note that:

$\mathcal{C}_1 = (\uparrow 0.7 : p_1) \quad \mathcal{C}_2 = (\uparrow 0.6 : p_3) \quad \mathcal{C}_3 = (\uparrow 0.8 : p_6)$

$\mathcal{C}_4 = (\{\downarrow 0.2 : p_1 \wedge \downarrow 0.1 : p_2\} \vee \{\downarrow 0.15 : p_3\} \vee \{(\downarrow 0.25 : p_4 \vee \downarrow 0.4 : p_5) \wedge$
$(\downarrow 0.2 : p_6)\} \vee \{\uparrow 0.8 : p_7 \wedge \uparrow 0.7 : p_8\})$

$\mathcal{C}_5 = (\{(\downarrow 0.3 : p_1 \vee \downarrow 0.3 : p_7) \wedge (\downarrow 0.2 : p_9)\} \vee \{\uparrow 0.2 : p_8 \wedge \uparrow 0.8 : p_9\})$

$\mathcal{C}_6 = (\{\downarrow 0.1 : p_8\})$

A proof sequence of the unsatisfiability of this formula Γ *is the following:*

$\mathcal{C}_1 = (\uparrow 0.7 : p_1)$,
$\mathcal{C}_4 = (\{\downarrow 0.2 : p_1 \wedge \downarrow 0.1 : p_2\} \vee \{\downarrow 0.15 : p_3\} \vee \{(\downarrow 0.25 : p_4 \vee \downarrow 0.4 : p_5) \wedge$
$(\downarrow 0.2 : p_6)\} \vee \{\uparrow 0.8 : p_7 \wedge \uparrow 0.7 : p_8\})$
$\vdash_{RDUR}$
$\mathcal{C}_7 = (\{\downarrow 0.15 : p_3\} \vee \{(\downarrow 0.25 : p_4 \vee \downarrow 0.4 : p_5) \wedge (\downarrow 0.2 : p_6)\} \vee \{\uparrow 0.8 : p_7 \wedge \uparrow 0.7 : p_8\})$

$\mathcal{C}_2 = (\uparrow 0.6 : p_3)$,
$\mathcal{C}_7 = (\{\downarrow 0.15 : p_3\} \vee \{(\downarrow 0.25 : p_4 \vee \downarrow 0.4 : p_5) \wedge (\downarrow 0.2 : p_6)\} \vee \{\uparrow 0.8 : p_7 \wedge \uparrow 0.7 : p_8\})$
$\vdash_{RDUR} \mathcal{C}_8 = (\{(\downarrow 0.25 : p_4 \vee \downarrow 0.4 : p_5) \wedge (\downarrow 0.2 : p_6)\} \vee \{\uparrow 0.8 : p_7 \wedge \uparrow 0.7 : p_8\})$

$\mathcal{C}_3 = (\uparrow 0.8 : p_6)$,
$\mathcal{C}_8 = (\{(\downarrow 0.25 : p_4 \vee \downarrow 0.4 : p_5) \wedge (\downarrow 0.2 : p_6)\} \vee \{\uparrow 0.8 : p_7 \wedge \uparrow 0.7 : p_8\})$
$\vdash_{RCUR} \mathcal{C}_9 = (\{\uparrow 0.8 : p_7 \wedge \uparrow 0.7 : p_8\})$

$$\frac{\mathcal{C}_9 = (\{\uparrow 0.8 : p_7 \wedge \uparrow 0.7 : p_8\})}{\mathcal{C}_{10} = (\uparrow 0.8 : p_7), \mathcal{C}_{11} = (\uparrow 0.7 : p_8)} (RAE)$$

$$\frac{\mathcal{C}_6 = (\{\downarrow 0.1 : p_8\})}{\mathcal{C}_{12} = (\downarrow 0.1 : p_8)} (RAE)$$

$$\frac{\mathcal{C}_{11} = (\uparrow 0.7 : p_8), \mathcal{C}_{12} = (\downarrow 0.1 : p_8)}{\mathcal{C}_{13} = \Box} (RDUR)$$

Theorem 5.4.1 Correctness *Let us LC={RSIMPLIF,RDUR,RCUR,RAE}.*

$$SRC(LC)$$

Or in other terms,

$$\Gamma \vdash_{LC} \Box \Longleftrightarrow UNSAT(\Gamma)$$

To prove the previous theorem, we have to prove the Soundness and Refutation Completeness of the defined Logical Calculus.

Lemma 5.4.1 Soundness

$$\Gamma \vdash_{RSIMPLIF} \mathcal{F} \Longrightarrow \Gamma \models \mathcal{F}$$

$$\Gamma \vdash_{RDUR} \mathcal{F} \Longrightarrow \Gamma \models \mathcal{F}$$

$$\Gamma \vdash_{RCUR} \mathcal{F} \Longrightarrow \Gamma \models \mathcal{F}$$

$$\Gamma \vdash_{RAE} (p) \Longrightarrow \Gamma \models (p)$$

Proof. The soundness of each rule of the Logical Calculus follows from the definitions of model and that of logical consequence. ■

Lemma 5.4.2 Refutation Completeness

$$UNSAT(\Gamma) \Longrightarrow \Gamma \vdash \Box$$

Proof. Base case

Assume the formula has no literals. The only formulas without literals are:

$$\Gamma = \{\}$$

and

$$\Gamma = \{\Gamma^1_\Box \wedge \Gamma^2_\Box \wedge \ldots \wedge \Gamma^k_\Box\}$$

where each clause $\Gamma^i_\Box$ is formed by the following three optional terms:

$$\Gamma^i_\Box = (\{() \wedge () \ldots \wedge ()\} \vee -\{\} \vee \{\} \vee \ldots \vee \{\} - \vee\{\})$$

Given that the $DNF^- = \{\} \vee \{\} \vee \ldots \vee \{\}$ term and the $C^+ = \{\}$ term are satisfiable ones, the only unsatisfiable formulas are those containing at least one clause $\Gamma^i_\Box$ without anyone of the three terms, $\Gamma^i_\Box = () \equiv \Box$, or only containing the CNF^- term $\Gamma^i_\Box = (\{() \wedge () \ldots \wedge ()\})$. In these both cases, $\Gamma^i_\Box$ is unsatisfiable and hence, the formula Γ is also unsatisfiable.

So, we need to prove the theorem only for these two cases of formulas.

1. Case $\Gamma^i_\Box = \Box$. $\Box$ is deduced by the reflexivity property of the inference relation $\vdash$.

2. Case $\Gamma^i_\Box = (\{() \wedge () \ldots \wedge ()\})$. In this case, applying the $RSIMPLIF$ rule, the clause $() \equiv \Box$ is deduced.

Let us prove the theorem for formulas with literals.

We note $\mathcal{F} = (C_1 \vee C_2 \vee \ldots \vee C_k \vee \{D_1 \wedge D_2 \wedge \ldots \wedge D_n\} \vee \mathcal{C}^+)$

notation. For readability, in this proof we will write C instead of C^- and D instead of D^-.

Let us $pos(p) = \{\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_k\}$ be the set of clauses such that $(\uparrow x_i : p) \in \mathcal{C}_i^+$, $1 \leq i \leq k$. We define the rank of a literal as follows:

If $\mathcal{F} = (\mathcal{C}^+)$ and $(\uparrow x : p) \in \mathcal{C}^+$ then $rank((\uparrow x : p)) = 1$.
If $(\downarrow y : p) \in \mathcal{F}$ and $pos(p) = \emptyset$ then $rank((\downarrow y : p)) = 1$.
Otherwise:

$$rank((\uparrow x : p)) = 1 + max\{rank((\downarrow y : p_i)) : (\downarrow y : p_i) \in \mathcal{F} \in pos(p)\}$$

$$rank((\downarrow x : p)) = max\{rank((\uparrow y : p)) : y \geq x, (\uparrow y : p) \in pos(p)\}$$

We note Γ^+ the set of clauses of Γ containing a positive conjunction C^+ and Γ^- the set of clauses of Γ containing exclusively negated literals.

Lemma 5.4.3 *If* $\Gamma^+ \models (\uparrow x' : p)$ *then there exists at least one clause* $\mathcal{F} = (C_1 \vee C_2 \vee \ldots \vee C_k \vee \{D_1 \wedge D_2 \wedge \ldots \wedge D_n\} \vee \mathcal{C}^+)$ *such that*

1. $(\uparrow x : p) \in \mathcal{C}^+$*, or similarly,* $\mathcal{F} \in pos(p)$*, with* $rank((\uparrow x : p)) = k$ *and* $x \geq x'$.
2. $\forall(\downarrow y : p_i) \in \mathcal{F}$, $rank((\downarrow y : p_i)) < k$
3. $\forall C_i$,$\exists(\downarrow y_i : p_i)$, $(\downarrow y_i : p_i) \in C_i$ *s.t.* $\Gamma^+ \models (\uparrow x_i : p_i)$ *with* $x_i > y_i$ *and* $\exists D_j$, $\forall(\downarrow y_j : p_j)$,$(\downarrow y_j : p_j) \in D_j$ *s.t.* $\Gamma^+ \models (\uparrow x_j : p_j)$ *with* $x_j > y_j$.

Proof.

1. Assume $\nexists \mathcal{F}, \mathcal{F} \in pos(p)$ and $x \geq x'$. Let us I be a model of Γ^+ such that $I(p) \geq x'$. Let us consider an interpretation I' with the same mapping that I except for $I'(p) = x' - \epsilon$ where ϵ is almost 0. Obviously I is also a model of Γ^+ because I' satisfies the same literals that I since all the occurrences of p in Γ^+ have less polarity than x'. Hence, $\Gamma^+ \not\models (\uparrow x' : p)$.
2. Assume that $\exists(\downarrow y : p_i)$ such that $rank((\downarrow y : p_i)) = l \geq k$. Then by definition of rank, $rank((\uparrow x : p)) = l + 1 > k$.
3. Assume that $\exists C_i$,$\forall(\downarrow y_i : p_i)$, $(\downarrow y_i : p_i) \in C_i$ such that $\Gamma^+ \not\models (\uparrow x_i : p_i)$ with $x_i > y_i$ or assume that $\forall D_j$, $\exists(\downarrow y_j : p_j)$, $\Gamma^+ \not\models (\uparrow x_j : p_j)$ with $x_j > y_j$. We provide the proof for the first alternative, the proof for the second one is completely similar. If $(\uparrow x_i : p_i)$ with $x_i > y_i$ is not a logical consequence

means that there is a model of Γ^+ such that $I(p_i) < x_i$, namely I satisfies $(\downarrow y_i : p_i)$ and then, C_i is satisfied by I. But, taking into account this fact, defining I' equal to I except for the mapping $I'(p) < x$ is also a model and then we have $\Gamma^+ \not\models (\uparrow x : p)$.

■

Lemma 5.4.4 $\Gamma^+ \models (\uparrow x' : p) \Longrightarrow \Gamma \vdash ((\uparrow x : p)),\ x \geq x'$.

Proof. By induction of the statement: If literals $(\uparrow x : p)$ of rank smaller than k are deduced then, literals of rank k are also deduced.

INDUCTION BASE: The statement is true for propositions of rank 1. By definition, the only literals $(\uparrow x : p)$ with rank 1 and such that $\Gamma \models (\uparrow x : p)$ are those that $(\uparrow x : p) \in \mathcal{C}^+ \in \Gamma$. Then applying $RAE(\mathcal{C}^+)$ we produce $((\uparrow x : p))$ because $(\uparrow x : p) \in \mathcal{C}^+$.

INDUCTION HYPOTHESIS: Assume that the induction statement is true for literals of rank smaller than k and consider that $(\uparrow x : p)$ is of rank k. Then, by previous lemma there exists at least one clause
$\mathcal{F} = (C_1 \vee C_2 \vee \ldots \vee C_k \vee \{D_1 \wedge D_2 \wedge \ldots \wedge D_n\} \vee \mathcal{C}^+)$ such that

1. $(\uparrow x : p) \in \mathcal{C}^+$, or similarly, $\mathcal{F} \in pos(p)$.
2. $\forall (\downarrow y : p_i) \in \mathcal{F}$, $rank((\downarrow y : p_i)) < k$.
3. $\forall C_i, \exists (\downarrow y_i : p_i)$, $(\downarrow y_i : p_i) \in C_i$ s.t. $\Gamma^+ \models (\uparrow x'_i : p_i)$ with $x'_i > y_i$ and $\exists D_j$, $\forall (\downarrow y_j : p_j), (\downarrow y_j : p_j) \in D_j$ s.t. $\Gamma^+ \models (\uparrow x'_j : p_j)$ with $x'_j > y_j$.

Then, by induction hypothesis, $\Gamma^+ \vdash ((\uparrow x_i : p_i))$ with $x_i \geq x'_i$ and $\Gamma^+ \vdash ((\uparrow x_j : p_j))$ with $x_j \geq x'_j$. Applying iteratively the RCUR rule upon $\mathcal{F}$ and each $(\uparrow x_i : p_i)$ we obtain $\mathcal{F} = (\{D_1 \wedge D_2 \wedge \ldots \wedge D_n\} \vee \mathcal{C}^+)$. Next, applying iteratively the RDUR rule upon $\mathcal{F}$ and each $(\uparrow x_j : p_j)$ we obtain $\mathcal{F} = (\{D_1 \wedge D_2 \wedge \ldots \wedge D_{j-1} \wedge \Box \wedge D_{j+1} \wedge \ldots \wedge D_n\} \vee \mathcal{C}^+)$. Next, applying RSIMPLIF rule upon $\mathcal{F}$, we deduce $(\mathcal{C}^+)$ and followed by a RAE rule we derive $((\uparrow x : p))$ because $(\uparrow x : p) \in \mathcal{C}^+$. ■

Lemma 5.4.5 *$UNSAT(\Gamma) \Longrightarrow \exists \mathcal{F}^-, \mathcal{F}^- \in \Gamma$ such that:*

1. *$\mathcal{F}^- = (C_1 \vee C_2 \vee \ldots \vee C_k \vee \{D_1 \wedge D_2 \wedge \ldots \wedge D_n\})$*
2. *$\forall C_i$, $\exists (\downarrow y_i : p_i)$, $(\downarrow y_i : p_i) \in C_i$ s.t. $\Gamma^+ \models (\uparrow x_i : p_i)$, $x_i > y_i$*
3. *$\exists D_j$, $\forall (\downarrow y_j : p_j), (\downarrow y_j : p_j) \in D_j$ s.t. $\Gamma^+ \models (\uparrow x_j : p_j)$, $x_j > y_j$*

Proof.

1. By definition of Γ^-, $\mathcal{F}^-$ has the correct structure.

2. The proof is by contradiction. Assume $\exists C_i, \forall(\downarrow y_i : p_i), (\downarrow y_i : p_i) \in C_i$ s.t. $\Gamma^+ \not\models (\uparrow x_i : p_i)$ with $x_i > y_i$. Take a model I of Γ^+ such that $I(p_i) < x_i$. Such a model exists because by hypothesis $\Gamma^+ \not\models (\uparrow x_i : p_i)$. We have that C_i is satisfied by I and then $\mathcal{F}^-$ and Γ^- are also satisfied and as I is a model of Γ^+ it turns out that Γ is satisfiable.

3. Assume $\forall D_j$, $\exists(\downarrow y_j : p_j), (\downarrow y_j : p_j) \in D_j$ s.t. $\Gamma^+ \not\models (\uparrow x_j : p_j)$. The proof of this case is similar to the previous case.

■

Lemma 5.4.6 $UNSAT(\Gamma) \Longrightarrow \Gamma \vdash \Box$

Proof. By the previous lemma, we have

1. $\Gamma^+ \models (\uparrow x_i : p_i)$
2. $\Gamma^+ \models (\uparrow x_j : p_j)$

and, by the lemma 5.4.4, we have

1. $\Gamma^+ \vdash ((\uparrow x'_i : p_i))$, $x'_i \geq x_i$
2. $\Gamma^+ \vdash ((\uparrow x'_j : p_j))$, $x'_j \geq x_j$

Applying iteratively the RCUR rule upon each $(\uparrow x'_i : p_i)$ and $\mathcal{F}$ we get $\mathcal{F}^- = (\{D_1 \wedge D_2 \wedge \ldots \wedge D_n\})$. Now applying iteratively the RDUR rule upon each $(\uparrow x'_j : p_j)$ and $\mathcal{F}^-$ we get $\mathcal{F}^- = (\{D_1 \wedge D_2 \wedge \ldots \wedge D_{j-1} \wedge \Box \wedge D_{j+1} \wedge \ldots \wedge D_n\})$. Then, applying the RSIMPLIF upon $\mathcal{F}$ we obtain (), that is logically equivalent by definition to $\Box$. ■

As the previous lemma is the same that the completeness theorem, the proof of this last lemma concludes the proof of the completeness theorem ■

This proves **MAIN THEOREM 1** of our proof methodology.

5.4.3 Logical Calculus: Its polynomial algorithmic version

In this section, we are going to prove the MAIN THEOREM 2 of our proof methodology:

MAIN THEOREM 2: $\exists x, CORSAT(x) \wedge PSAT(x)$

A proof of the unsatisfiability of a Regular SIMPLE-HORN-NNF formula is constructed by applying consecutively inferences rules, as it has been indicated in the definition of Refutation Proof and in the subsequent example. Applying this process of inference rules sequencing does not lead to an efficient SAT algorithm. One of the original causes is because new sub-formulas of the original formula, are generated and then added to the original formula. Thus, the formula obtained after a certain number of inferences is the original formula augmented with copies of some of their sub-formulas. However, it is easy to prove that the upper-bound complexity of a SAT algorithm implementing the Logical Calculus is Polynomial.

Definition 5.4.15 *Let us define $SATALG_P$, an algorithm that implements the defined Logical Calculus as follows:*

1. *It choices the inferences to be applied according to a deterministic criterion*
2. *It returns "UNSAT" iff the defined Logical Calculus derives $\Box$.*

More specifically, the algorithm scheme is the following, where Γ is the initial set of subformulas.

```
$SATALG_P(\Gamma)$
    While the set of executables inference rules is not empty do:
      $\forall (C^+) \in \Gamma$ do: $\forall (\uparrow i : p_i) \in (C^+) \in \Gamma$: add $(\uparrow i : p_i)$ to $\Gamma$
      Execute applicable RDUR rules adding the deduced subformulas to $\Gamma$
      Execute applicable RCUR rules adding the deduced subformulas to $\Gamma$
      Execute applicable RSIMPLIF rules adding the deduced subformulas to $\Gamma$
    EndWhile
    If $() \in \Gamma$ then return("UNSAT") Else return("SAT")
```

Theorem 5.4.2 $CORSAT(SATALG_P) \wedge PSAT(SATALG_P)$.

Proof. It is trivial that the previous algorithm is sound because it executes only rules that are in the original Logical Calculus. Now let us prove $LCSATALG(LC, SATALG_P)$. Concerning the completeness of the algorithm, one can check that the difference with respect to the Logical Calculus is that in the algorithm the inference rules are executed in a certain order. Aiming at proving $LCSATALG(LC, SATALG_P)$, we need to prove that the order followed by the algorithm preserves the completeness of the Logical Calculus. To this end, we need to prove that if we execute first a certain inference rule the applicable inference rules before such execution remain applicable after the execution. For instance, assume that we execute first a RDUR rule. Thus, if we execute first a RDUR rule with (p) and $\mathcal{F}$ the resulting subformula $\mathcal{F}_p$ is added to Γ and no subformula is eliminated from Γ. Thus, inference rules applicable before the execution of the RDUR rule remain applicable after executing it. The same reasoning can be done when the first inference executed is RCUR, RSIMPLIF or RAE. Therefore, this proves $LCSATALG(LC, SATALG_P)$ and this, together with $SRC(LC)$ and corollary 5.3.2, leads to $CORSAT(SATALG_P)$. Now, we shall prove $PSAT(SATALG_P)$.

1. It could be checked that the maximal number of executable RDUR and RCUR inference rules is bounded by the number n of proposition occurrences in the original formula because each time that a RDUR or a RCUR is applied, a negative occurrence of a proposition or a whole conjunction is removed.
2. As the size of the inferred sub-formula $\mathcal{F}_i$ is bounded by the size of the original formula $F_0 = \Gamma$ in an inference rule, we have:

$$< F_0, F_1, \ldots, F_n >, \text{ with } F_0 = \Gamma, F_{i+1} = F_i \wedge \mathcal{F}_i, F_n = F_{n-1} \wedge \Box,$$

where:

$$size(F_0) = n$$

$$size(F_1) = size(F_0) + size(\mathcal{F}_0) < size(F_0) + size(F_0) = 2.n$$

$$size(F_i) = size(F_0) + size(\mathcal{F}_0) + size(\mathcal{F}_1) + \ldots + size(\mathcal{F}_{i-1}) <$$

$$size(F_0) + size(F_0) + size(F_0) + \ldots + size(F_0) <$$

$$size(F_i) < (i+1).n$$

$$size(F_n) \in O(n^2)$$

$$\forall i, 1 \leq i \leq n, size(F_i) \in O(n^2)$$

In complexity terms, the length of an inferred formula created by $SATALG_P$ increases in $O(n^2)$.

3. On the other hand, as the inferred formula increases in $O(n^2)$, the search time for the two involved clauses in an inference rule requires to scan the current inferred formula and hence, it is in $O(n^2)$.

4. Having searched the two involved clauses, executing with them an inference step can be done in a time bounded by $O(n)$, the time required to copy a sub-formula $\mathcal{F}_i$.

5. Then, searching and executing one inference rule is limited by $O(n^2)$.

6. As the number of inferences is bounded by $size(F_0) = n$, the total complexity of the construction of an **Refutation proof** in $SATALG_P$ is in $O(n^3)$.

7. Therefore, that proves that the Regular SIMPLE-HORN-NNF SAT problem is in $\mathcal{P}$.

■

Altogether, a straight computer implementation in a SAT algorithm $SATALG_P$, of the generation of **Refutation Proofs** corresponding to the defined Logical Calculus, is in $O(n^3)$. This proves the MAIN THEOREM 2, i.e. that the SIMPLE-HORN-NNF SAT problem is polynomial.

5.4.4 A quadratic algorithm

Now, in this section we are going to prove the third theorem. This means that we will try to find a quadratic algorithm to resolve the Regular SIMPLE-HORN-NNF SAT problem. We will do it in several optimization steps.

A first optimization, is to avoid multiple copies of subformulas. To do that, we rewrite the logical calculus in an "algorithmic way" by the redefinition of each inference rule in such a way that, each rule will be applied over the input formula eliminating symbols and, in this way, the original formula will reduce sequentially its original size. As it can be checked observing the inference rules, the inferred formula (the consequent part) is formed from the original one (antecedent part) by removing subformulas, i.e. literals or conjunctions in the input formula. This fact allows us to redefine the inference rules by making explicit which subformulas should be eliminated.

Quadratic Algorithmic Version of the Logical Calculus (QAVLC)

Definition 5.4.16 *Let us $(\downarrow j : p)$ be a disjunct in D^- which in turn belongs to subformula Γ_x. We note $RemoveLiteral((\downarrow j : p), D^-, \Gamma_x)$, the algorithmic function that removes physically the occurrence of the literal $(\downarrow j : p)$ in D^- with $i > j$ from Γ_x by acting directly on the data structure representing the subformula Γ_x.*

Definition 5.4.17 *Similarly to the above definition, let us C^- be a disjunct in a DNF^- term which in turn belongs to a subformula Γ_x. We note $RemoveConjunc tion(C^-, DNF^-, \Gamma_x))$ the algorithmic function that removes physically the occurrence of the conjunction C^- in the DNF^- term from Γ_x by acting directly in the data structure representing the subformula Γ_x. Also, $RemoveCNFterm(\Box, CNF^-, \Gamma_x)$ removes physically a CNF^- term of Γ_x if $\Box \in CNF^-$.*

Definition 5.4.18 *The new inference rules using functions Remove are named RCNF, RL, and RC for the previous $RSIMPLIF$, $RDUR$ and $RCUR$ functions, respectively, and they are defined as follows:*

$$\frac{\Gamma_x = (CNF^- = \{D_1^- \wedge \ldots \wedge \Box \wedge \ldots \wedge D_k^-\} \vee DNF^- \vee C^+)}{\Gamma_x = RemoveCNFterm(\Box, CNF^-, \Gamma_i)}(RCNF)$$

$$\frac{r_i > j_i, (\uparrow r_i : p_i), \atop \Gamma_x = (\{D_1^- \wedge \ldots \wedge D_j^- = (\downarrow j_1 : p_1 \vee \ldots \vee \downarrow j_i : p_i \vee \ldots \vee \downarrow j_n : p_n) \wedge \ldots \wedge D_s^-\} \vee DNF^- \vee C^+)}{\Gamma_x = RemoveLiteral((\downarrow j_i : p_i), D_j^-, \Gamma_x)}(RL)$$

$$\frac{r_i > j_i, (\uparrow r_i : p_i), \atop \Gamma_x = (CNF^- \vee DNF^- = [C_1^- \vee \ldots \vee C_j^- = \{\downarrow j_1 : p1 \wedge \ldots \wedge \downarrow j_i : p_i \wedge \ldots \wedge \downarrow j_n : p_n\} \vee \ldots \vee C_m^-] \vee C^+)}{\Gamma_x = RemoveConjunction(C_j^-, DNF^-, \Gamma_x)}(RC)$$

Corollary 5.4.1 *Let us $QAVLC = \{RCNF, RL, RC, RAE\}$, then $SRC(QAVLC)$.*

Proof. It is similar to the proof of corollary 4.4.1 in previous chapter. ■

Another step forward before obtaining an Algorithmic Version of the Logical Calculus consists in applying RL with $(\uparrow i : p)$, as many times as there are occurrences of $(\downarrow j : p)$, and such that $i > j$ exists in D_k^- disjunctions integrated in a CNF term $\{D_1 \wedge D_2^- \wedge \ldots \wedge D_m^-\}$ of any subformula Γ_x. Afterwards, the RC follows with the same principle: removing all the conjunctions C^- containing $(\downarrow j : p)$ such that $i > j$ from a subformula Γ_x

Definition 5.4.19 *Let us $(\downarrow j : p)$ be a disjunct in D^- (resp. conjunct in C^-) which in turn is in a CNF^- (resp. DNF^-) term. We note $RemAllLitSubForm((\uparrow i : p), \Gamma_x)$ (resp. $RemAllConjSubForm((\uparrow i : p), \Gamma_x)$) the algorithmic function that removes physically all the occurrences of $(\downarrow j : p)$ literals (resp. conjunctions containing a $(\downarrow j : p)$ occurrence) with $i > j$ from Γ_x, by acting directly on the data structure representing the subformula Γ_x.*

This new function leads to new inference rules.

Definition 5.4.20 *The new inference rules using $RemAllLitSubForm$ and $RemAllConjSubForm$ functions are called $RALSF$ and $RACSF$ and they are defined as follows:*

$$\frac{(\uparrow i : p), \Gamma_x = (CNF^- \vee DNF^- \vee C^+)}{\Gamma_x = RemAllLitSubForm((\uparrow i : p), \Gamma_x)}(RALSF)$$

$$\frac{(\uparrow i : p), \Gamma_x = (CNF^- \vee DNF^- \vee C^+)}{\Gamma_x = RemAllConjSubForm((\uparrow i : p), \Gamma_x)}(RACSF)$$

Corollary 5.4.2 *$RALSF$ and $RACSF$ are sound.*

Proof. It is similar to the proof of corollary 4.4.2 in previous chapter. ■

Definition 5.4.21 Quadratic Algorithmic Version of the Logical Calculus (QAVLC) *We redefine the Quadratic Algorithmic Version of the Logical Calculus as the set of inference rules formed by: $QAVLC = \{RCNF, RALSF, RACSF, RAE\}$.*

Theorem 5.4.3 Correctness $\forall \Gamma, \Gamma \vdash_{QAVLC} \Box \Longleftrightarrow UNSAT(\Gamma)$

Lemma 5.4.7 Soundness $\forall \Gamma, \Gamma \vdash_{QAVLC} \Box \Longrightarrow UNSAT(\Gamma)$.

Proof. RCNF and RAE are the same rules as previously defined in 5.4.18 and 5.4.13 respectively. On the other hand, the soundness of $RALSF$ and $RACSF$ have been established in the last two corollaries. ■

Lemma 5.4.8 Refutation Completeness $UNSAT(\Gamma) \Rightarrow \Gamma \vdash_{QAVLC} \Box$

Proof. It is very similar to the proof of lemma 4.4.8 in previous chapter. ■

The definitive algorithmic inferences derived from the Logical Calculus and which will serve to design an efficient SAT algorithm are the following ones.

Definition 5.4.22 *We note $RemoveAllLiterals((\uparrow i : p), \Gamma)$ (resp. $RemoveAll$ $Conjunctions((\uparrow i : p), \Gamma)$) the algorithmic function that removes physically all the occurrences of literals $(\downarrow j : p)$ (resp. conjunctions containing a $(\downarrow j : p)$ occurrence) with $i > j$ from the formula Γ, by acting directly on the data structure representing Γ. The algorithmic function $RemoveAllCNFs$ removes physically all the CNF terms containing the empty clause $\Box$.*

These new functions lead to new inference rules.

Definition 5.4.23 *The new inference rules are called RAL, RAC and $RACNF$ for $RemoveAllLiterals$, $RemoveAllConjunctions$, $RemoveAllCNFs$ and they are defined as follows:*

$$\frac{(\uparrow i : p), \Gamma}{\Gamma = RemoveAllLiterals((\uparrow i : p), \Gamma)}(RAL)$$

$$\frac{(\uparrow i : p), \Gamma}{\Gamma = RemoveAllConjunctions((\uparrow i : p), \Gamma)}(RAC)$$

$$\frac{\Gamma}{\Gamma = RemoveAllCNFs(\Box, \Gamma))}(RACNF)$$

Theorem 5.4.4 Correctness. *Let us QAVLC=$\{RAL, RAC, RACNF, RAE\}$.*

$$\forall \Gamma, \Gamma \vdash_{QAVLC} \Box \Longleftrightarrow UNSAT(\Gamma)$$

Proof. The proof is straightforward from the previous correctness of theorem 5.4.3. ■

A Correct Quadratic Algorithm

Once established the Algorithmic Version of the Logical Calculus, we can design a first algorithm which is a strict materialization of the mechanization of the inference rules in $QAVLC$ (last definition).

When all subformulas Γ_x of Γ contain a negative subformula NNF^- then Γ is trivially satisfiable: a model is obtained by assigning the value 0 to all propositional variables. So assume that some positive conjunctions C^+ are present in Γ. Thus, applying the RAE rule over these conjunctions produces unit clauses $(\uparrow i : p)$. Altogether, this leads to the statement that the formula Γ is satisfiable or otherwise, some unit clauses can be deduced.

Thus, the next step is applying the inference rules in $QAVLC$ defined in 5.4.4 with unit clauses. This process is repeated until either no more unit clauses are generated or an empty clause is produced. In the first case, the formula is satisfiable and in the second one, it is unsatisfiable.

The principle of the algorithm is the following. First, the RAE inference rule is applied to positive conjunctions C^+ giving as result several unit clauses which are then pushed in a stack (function $ApplyRAE(\Gamma, Stack)$). For each proposition in the $Stack$, the rules RAL, RAC and $RACNF$ are applied throughout the respective algorithmic functions $RemoveAllLiterals$, $RemoveAllConjunctions$ and $RemoveAllCNFs$. If as a consequence of the $Remove$ functions, some subformulas become positive conjunctions C^+, the RAE rule is applied upon C^+ adding new propositions to the $Stack$. This process finishes when the $Stack$ becomes empty, which means that there is no more inferences to apply. Thus, if in that situation the empty clause has not been deduced yet, then the input formula Γ is satisfiable and else is unsatisfiable.

```
$SATALG_{Q1}(\Gamma)$
1 ApplyRAE($\Gamma$, Stack)
2 While $Stack \neq \{\}$ do:
3    $(\uparrow i : p) \leftarrow pop(Stack)$
4    RemoveAllLiterals($(\uparrow i : p), \Gamma$)
5    RemoveAllCNFs($\Box, \Gamma$)
6    RemoveAllConjunctions($(\uparrow i : p), \Gamma$)
7    ApplyRAE($\Gamma$, Stack)
8 EndWhile
9 If $() = \Gamma_x \in \Gamma$ then return("UNSAT") Else return("SAT")
```

Theorem 5.4.5 Correctness *$SATALG_{Q1}(\Gamma)$ returns "UNSAT" iff Γ is unsatisfiable.*

Proof. This theorem is a direct consequence of the Soundness and Refutation Completeness of the QAVLC, stated in theorem 5.4.4.

Indeed, we have the following parallelism. The line 1 starts the executions of inferences with RAE. The lines 4,5, 6 and 7 correspond respectively to the applications of the RAL, $RACNF$, RAC and RAE rules. The order of application of the inferences does not prevent completeness (see theorem 5.4.2).

When the line 9 is executed, the "while" iteration is finished. But the condition for termination of the "while" loop is verified when there is no more inference applicable. So, as indicated in line 9, Γ is satisfiable iff the empty clause has not been deduced. So, we have $LCSATALG(QAVLC, SATALG_{Q1})$. The proof continues as follows:

1. $\forall x, y, LCSATALG(x, y) \wedge SRC(x) \Longrightarrow CORSAT(y)$, corollary 4.3.1
2. $LCSATALG(QAVLC, SATALG_{Q1})$, by the construction of $SATALG_{Q1}$,
3. $SRC(QAVLC)$, proved in 5.4.4.

We obtain: $CORSAT(SATALG_{Q1})$. ■

Theorem 5.4.6 *The complexity of the algorithm $SATALG_{Q1}$ is $O(n^2)$.*

Proof. The execution of each line 4, 5, 6 and 7 takes a time bounded by $O(n)$. As the number maximum of iterations is also bounded by $\Sigma_{C^+ \in \Gamma} |C^+| < O(n)$, therefore the complexity is in $O(n^2)$. ■

Our next goal is to design a data structure for the previous Algorithmic Schema and to prove that an almost linear algorithm can be derived from it.

5.4.5 An almost linear Algorithm

Before describing the final algorithm, we describe first a preliminary and almost linear **incorrect** algorithm containing all the data structures (except one subtle data structure to be presented later). This is done as a previous steps in order to help to understand the final correct polynomial SAT algorithm. So, firstly we describe the required data structure and then, we give the corresponding algorithms.

Data Structure. To each proposition p_k, we associate two lists of pointers $D^-(k)$ and $C^-(k)$. Each element (i,j) in $D^-(k)$ (resp. $C^-(k)$) is a pointer to a disjunction D_j^- (resp. conjunction C_j^-) in Γ_i. A couple (i,j) in $D^-(k)$ (resp. $C^-(k)$) means that $(\downarrow k : p_k) \in D_i^- \in \Gamma_j$ (resp. $(\downarrow k : p_k) \in C_i^- \in \Gamma_j$). For each subformula Γ_i, we note $C^+(i)$ the list of positive propositions in the C^+ term of Γ_i.

Input: $\Gamma = \Gamma_1 \wedge \Gamma_2 \wedge \ldots \wedge \Gamma_n$
Output: "SAT" or "UNSAT".

Initialization procedure. In this step, all the data structure, i.e. $D^-(k)$, $C^-(k)$ and $C^+(i)$, are initialized according to their definition and the input formula Γ.

```
$SATALG_{Q2}(\Gamma)$
1 $\forall \Gamma_i = (C^+)$ do: $\forall (\uparrow x : p) \in C^+$ do: $push((\uparrow x : p), Stack)$
2 While $Stack \neq \{\}$ do:
3     $(\uparrow x : p_k) \leftarrow pop(Stack)$
4     $\forall (i,j) \in D^-(k)$ do: $Remove((\downarrow y : p), D_{i,j})$ if $x > y$
5     $\forall (i,j) \in D^-(k)$ do: If $D_{i,j} = \emptyset$ then $Remove(CNF^- i, \Gamma_i)$
6     $\forall (i,j) \in C^-(k)$ do: $Remove((\downarrow y : p), C_{i,j})$ if $x > y$
7     $\forall (i,j) \in D^-(k) \bigcup C^-(k)$ do: If $\Gamma_i = C_i^+$: $\forall (\uparrow x : p) \in C^+(i)$ do: $push((\uparrow x : p), Stack)$
8 Endwhile
9 $\forall i, 1 \leq i \leq n$, do: If $\Gamma_i = ()$ then return("UNSAT") Else return("SAT")
```

Theorem 5.4.7 *$SATALG_{Q2}(\Gamma)$ returns "SAT" iff Γ is satisfiable.*

Proof. The parallelism between the instructions of this algorithm and those of the previous one is straightforward. For example, the previous line:

4 RemoveAllLiterals$((\uparrow i : p), \Gamma)$.

is substituted by the current line:

$$4\ \forall (i,j) \in D^-(k) \text{ do: } \mathit{Remove}((\downarrow y : p), D_{i,j}) \text{ if } x > y.$$

Both lines have the same effects: removing occurrences of $(\downarrow y : p)$ from disjunctions in the input formula Γ. In the former this statement is clearly verified. In the latter, one can check that the occurrences of $(\downarrow y : p)$ are exactly those indicated by list $D^-(k)$, and line 4 removes exactly these occurrences. The same analysis can be done to prove the algorithmic equivalence of lines 5, 6, and 7 of the previous algorithm and the current one which leads to the proof of the theorem. Then, we have:

1. $\forall x, y, SATEQU(x, y) \wedge CORSAT(x) \Longrightarrow CORSAT(y)$, corollary 5.3.2
2. $SATEQU(SATALG_{Q1}, SATALG_{Q2})$, by the construction of both algorithms
3. $CORSAT(SATALG_{Q1})$, by previous theorem 5.4.5

and hence, we obtain $CORSAT(SATALG_{Q2})$ ■

We can check that in the previous algorithm the Remove functions have a $O(n)$ complexity because a subformula must be scanned. This complexity can be improved to obtain a $O(1)$ complexity for removing the pertinent element of the subformula. This is achieved by introducing two counters in the data structure.

Remark Notice that we do not need to know exactly which literal occurrences (resp. conjunctions) have been removed. What we need to know is merely how many elements have been removed in order to detect when a disjunction (resp. a whole CNF^- term) has been removed, i.e. has been falsified.

- **RALs: removing $(\downarrow y : p)$ occurrences.** To each disjunction D_j^- in clause Γ_i, a counter $\mathit{Counter}(i, j)$ is associated. A decrement of $\mathit{Counter}(i, j)$ indicates the removal of a falsified literal $(\downarrow y : p)$ in D_j^-.
- **RACNFs: removing CNF^- terms.** If any $\mathit{counter}(i, j)$ is set to 0 means that the disjunction D_j^- in Γ_i is falsified and hence the whole CNF^- term is also falsified. This is implemented by setting to 0 a flag $CNF(j)$ associated with each clause Γ_j.
- **Removes the DNF term.** A counter $\mathit{Counter.DNF}(i)$ is associated with the DNF^- term in each clause Γ_i. Each decrement of $\mathit{Counter.DNF}(i)$ represents a removal of a falsified conjunction C^- in the DNF^- term.

With these new data structures, the corresponding algorithm is:
Input: $\Gamma = \Gamma_1 \wedge \Gamma_2 \wedge \ldots \Gamma_n$
Output: "SAT" or "UNSAT".

Initialization procedure. In addition to the previous data structure ($D^-(k)$, $C^-(k)$ and $C^+(i)$), $Counter(i,j)$ and $Counter.DNF(i)$ are initialized according to their definition. A further data structure is required: $T(i,j,k)$ is a table that stocks the sign y of literal $(\downarrow y : p_k)$ occurring in position (i,j).

```
$SATALG_{AL}(\Gamma)$
1  $\forall\Gamma_i = (C^+)$ do: $\forall(\uparrow x : p) \in C^+$ do:$push((\uparrow x : p), Stack)$
2  While $Stack \neq \{\}$ do:
3      $(\uparrow x : p_k) \leftarrow pop(Stack)$
4      $\forall(i,j) \in D^-(k)$ do: Decrement $Counter(i,j)$ if $T(i,j,k) \leq x$
5      $\forall(i,j) \in D^-(k)$ do: If $Counter(i,j) = 0$ then $CNF^-(i) \leftarrow 0$
6      $\forall(i,j) \in C^-(k)$ do: Decrement $Counter.DNF^-(i)$ if $T(i,j,k) \leq x$
7      $\forall(i,j) \in (D^-(k) \bigcup C^-(p))$ and $CNF^-(i) = Counter.DNF(i) = 0$ do:
           $\forall(\uparrow x : p) \in C^+(i)$ do: $push((\uparrow x : p), Stack)$
8  Endwhile
9  $\forall i, 1 \leq i \leq n$, do:
10     If $CNF(i) = Counter.DNF(i) = C^+(k) = 0$ then return("UNSAT")
11     Else return("SAT")
```

The structure of the last algorithm is the same that the previous one. The sequence of applications of the inference rules is strictly the same in both algorithms. The parallelism between the operations performed by this algorithm and those of the previous Algorithmic Structure is straightforward.

However, the previous algorithm is not correct. Indeed, when we applied the inference by removing elements, there were not exist the problem of modifying erroneously the data structure representing the deduced formula. However, with the counters, that can happen because as the application of the inferences is done via the decrements of counters, for a same unit clause (p) a counter can be decremented several times. Actually, we have two types of errors in the previous algorithm:

- Given that a same proposition p can be present in several C^+ terms of the clauses Γ_i, each proposition p can be introduced in the stack several times. To avoid to execute several times the inference rules with the same proposition, we have to change the first two iterations. Thus, we organize the $D^-(k)$ and $C^-(k)$ sorting the occurrences (i,j) in incremental order, namely (i,j) is before (i',j'), in the lists $D^-(k)$ or $C^-(k)$, if $T(i,j,k) \leq T(i',j',k)$. This is explained in detail in section 2.3.3. After $(\uparrow x : p_k)$ is deduced all the occurrences of $(p_k, T(i,j,k))$ such that $x \geq T(i,j,k)$ are removed. The removing of couples (i,j) is identic to the remove of literals from the formula and that ensures that an inference is not repeated applying it to the same couples of literals.

  The two lines of the previous algorithm:

  4 $\forall(i,j) \in D^-(k)$ do: Decrement $Counter(i,j)$ if $T(i,j,k) < x$
  5 $\forall(i,j) \in D^-(k)$ do: If $Counter(i,j) = 0$ then $CNF^-(i) \leftarrow 0$

are substituted by an While-EndWhile iteration in the next algorithm as follows:

```
7     $(i,j) \leftarrow First(D^-(k))$
8     $y \leftarrow T(i,j,k)$
9     While $(x > y)$ do:
10        Decrement $Counter(i,j)$
11        If $Counter(i,j) = 0$ then $CNF^-(j) \leftarrow 0$
12        $Set1 \leftarrow Set1 \cup \{(i,j)\}$
13        Remove $First(D^-(k))$ from $D^-(k)$
14        $(i,j) \leftarrow First(D^-(k))$
15        $y \leftarrow T(i,j,k)$
16    Endwhile
```

Similarly, the line 6 of the previous algorithm:

6 $\forall (i,j) \in C^-(k)$ do: Decrement $Counter.DNF^-(i)$ if $T(i,j,k) \leq x$

is substituted by another End-While iteration with the same operations that the previous one:

```
17    $(i,j) \leftarrow First(C^-(k))$
18    $y \leftarrow T(i,j,k)$
19    While $(x > y)$ do:
20        If $First(i,j) = 1$ then do:
21            Decrement $Counter.DNF^-(i)$
22            $First(i,j) \leftarrow 0$
23        $Set2 \leftarrow Set2 \cup \{(i,j)\}$
24        Remove $First(C^-(k))$ from $C^-(k)$
25        $(i,j) \leftarrow First(C^-(k))$
26        $y \leftarrow T(i,j,k)$
27    Endwhile
```

- The second problem comes from the decrements of $Counter.DNF(i)$. Each decrement must correspond to the removal (i.e. falsification) of one conjunct of the DNF^- term. This counter should be set to 0 only when all the conjuncts are falsified. Nevertheless, in the previous algorithm, the deductions of n literals that could belong to a same conjunct C^- of the DNF_i^- term, implies n decrements of $Counter.DNF(i)$. Thus, the counter could be set to 0, indicating that the DNF^- term has been removed, in cases where not all the conjuncts in the DNF^- have been falsified. To overcome this problem, we use a flag call $First(C^-)$ for each conjunct C^- in the DNF^-. Initially, this flag is set to 1. After the first falsification of any literal in C^-, the flag is set to 0. In this way, only one decrement of $Counter.DNF$ is allowed for each conjunct C^- in DNF^-.

Thus, correcting the previous algorithm with the previous defined data structure, we have:

```
If $First(i,j)$= 1 then do: decrement $Counter.DNF(i)$
                            $First(i,j) \leftarrow 0$
```

The resulting algorithm is therefore:

```
$SATALG_{AL}(\Gamma)$
1  $\forall(C^+) \in \Gamma$ do: $\forall(\uparrow x : p) \in C^+$ do: $push((\uparrow x : p), Stack)$
2  While $Stack \neq \{\}$ do:
3      $(\uparrow x : p_k) \leftarrow pop(Stack)$
4      If $x > Val(p_k)$ then do:
5          $Val(p_k) \leftarrow x$
6          $Set1 \leftarrow \emptyset$; $Set2 \leftarrow \emptyset$
7          $(i, j) \leftarrow FirstElement(D^-(k))$
8          $y \leftarrow T(i, j, k)$
9          While $(x > y)$ do:
10             Decrement $Counter(i, j)$
11             If $Counter(i, j) = 0$ then $CNF^-(i) \leftarrow 0$
12             $Set1 \leftarrow Set1 \cup \{(i, j)\}$
13             Remove $FirstElement(D^-(k))$ from $D^-(k)$
14             $(i, j) \leftarrow FirstElement(D^-(k))$
15             $y \leftarrow T(i, j, k)$
16         Endwhile
17         $(i, j) \leftarrow FirstElement(C^-(k))$
18         $y \leftarrow T(i, j, k)$
19         While $(x > y)$ do:
20             If $First(i, j) = 1$ then do:
21                 Decrement $Counter.DNF^-(i)$
22                 $First(i, j) \leftarrow 0$
23             $Set2 \leftarrow Set2 \cup \{(i, j)\}$
24             Remove $FirstElement(C^-(k))$ from $C^-(k)$
25             $(i, j) \leftarrow FirstElement(C^-(k))$
26             $y \leftarrow T(i, j, k)$
27         Endwhile
28         $\forall(i, j) \in Set1 \bigcup Set2$ and $CNF(i) = Counter.DNF(i) = 0$ do:
29             $\forall(\uparrow x : p) \in C^+(i)$ do: $push((\uparrow x : p), Stack)$
30 Endwhile
31 $\forall i, 1 \leq i \leq n$, do:
32     If $CNF(i) = Counter.DNF(i) = C^+(i) = 0$ then return("UNSAT")
33     Else return("SAT")
```

Now, we can ensure the correctness of the algorithm:

Theorem 5.4.8 Correctness. *$SATALG_{AL}(\Gamma)$ returns "SAT" iff Γ is satisfiable.*

Proof. The parallelism between the instructions of this algorithm and those of the previous one is mentioned above. For example, lines in the algorithm 4, 5, y 6 are substituted by the two iterations specified.

Both lines 4, 5, and 6 and iterations While-EndWhile have the same effects: removing occurrences of $(\downarrow y : p)$ from disjunctions and conjunctions in the input formula Γ. In the former, this statement is clearly verified. In the latter, one can check that the occurrences of $(\downarrow y : p)$ are exactly those indicated by lists $D^-(k)$ and $C^-(k)$, and the iterations remove exactly these occurrences proceeding as it is detailed in section 2.3.3. Then, we have:

1. $\forall x, y, SATEQU(x, y) \wedge CORSAT(x) \Longrightarrow CORSAT(y)$, corollary 5.3.2
2. $SATEQU(SATALG_{Q2}, SATALG_{AL})$, by the construction of both algorithms
3. $CORSAT(SATALG_{Q2})$, by previous theorem 5.4.7

and hence, we obtain $CORSAT(SATALG_{AL})$ ■

Concerning the algorithms' complexity, the last algorithm is almost linear.

Theorem 5.4.9 Complexity *The algorithm $SATALG_{AL}$ is in $O(n.log(m))$, where n is the number of different propositions and m is the maximal number of negative occurrences of a proposition in the formula.*

Proof. The number of propositions inserted into the stack is bounded by the number of literals in positive conjunctions C_i^+ of subformulas Γ_i. The number of iterations in the While-Endwhile blocks is limited by the number of negative literals in the formula. Altogether, the number of operations is in $O(n)$. In the initialization procedure all the operations are linear except the operations of ordering D^- and C^- which are in $O(n.log(m))$, where where n is the number of different propositions and m is the maximal number of negative occurrences of a proposition in the formula. For more details about the calculus of this complexity, see section 2.3.3. ■

Following the methodology, we have proved the last theorem:

MAIN THEOREM 4: $\exists x, CORSAT(x), ALSAT(x)$

where x is the previous linear algorithm $SATALG_{AL}$

5.5 The Regular HORN-NNF-SAT problem

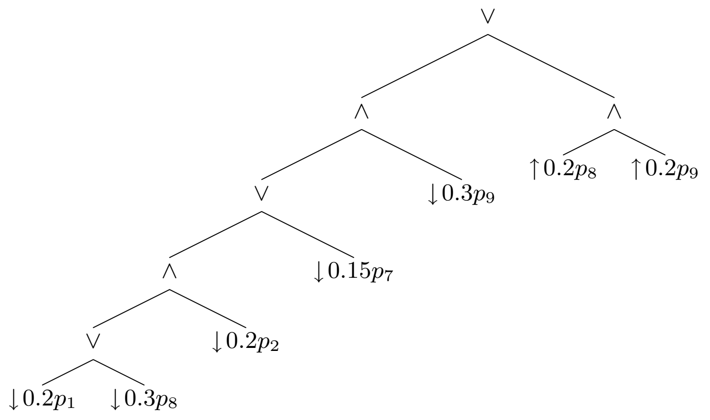
5.5.1 The Regular HORN-NNF formulas

In this section we define the syntax and semantic elements to the case of general Regular HORN-NNF formulas ending with the definition of the related SAT problem. We define those formulas as a generalization of the previous Regular SIMPLE-HORN-NNF formulas giving as a consequence that most of the definitions and related concepts are direct extensions of the previous ones defined in the corresponding section for the Regular SIMPLE-HORN-NNF formulas.

Definition 5.5.1 Regular HORN-NNF formulas *A Regular NNF formula is a regular formula formed by the well known induction over the set of regular literals* $\mathbf{L}$ *and the connectives* $\vee$ *and* $\wedge$*. A negative NNF formula, noted* NNF^-*, is an NNF formula formed exclusively by regular literals with negative polarity.*

Example 5.5.1 *The following formula is a Regular NNF formula.*

$$\begin{array}{c}
\Gamma = ((\uparrow 0.7 : p_1) \wedge (\uparrow 0.6 : p_3) \wedge (\uparrow 0.8 : p_6) \\
\wedge \\
(\{((\downarrow 0.2 : p_1) \vee (\downarrow 0.1 : p_8)) \wedge (\downarrow 0.15 : p_2)\} \quad \vee \quad (\downarrow 0.25 : p_3) \quad \vee \quad \{((\downarrow 0.25 : p_4) \vee \\
(\downarrow 0.4 : p_5)) \wedge (\downarrow 0.2 : p_6)\} \quad \vee \quad \{(\uparrow 0.3 : p_7) \wedge (\uparrow 0.7 : p_8)\}) \\
\wedge \\
((((((\downarrow 0.2 : p_1) \vee (\downarrow 0.3 : p_8)) \wedge (\downarrow 0.2 : p_2)) \vee (\downarrow 0.15 : p_7)) \wedge (\downarrow 0.3 : p_9)) \vee ((\uparrow 0.2 : p_8) \wedge (\uparrow 0.2 : p_9)) \\
\wedge \\
(\downarrow 0.1 : p_8))
\end{array}$$



Definition 5.5.2 Regular HORN-NNF formulas *A regular HORN-NNF clause is a regular clause formed by two optional terms: a regular* NNF^- *formula and a non-negative regular conjunction* C*:* $\mathcal{C} = NNF^- \vee C^+$*. Finally a Regular HORN-NNF formula is a set of Regular HORN-NNF clauses.*

The graphical representation of the fifth clause of this formula is given below following a straightforward extension of the previous principle. The conjunct (resp. disjunct) sub-formulas of a conjunction (resp. disjunction) are represented vertically (resp. horizontally).

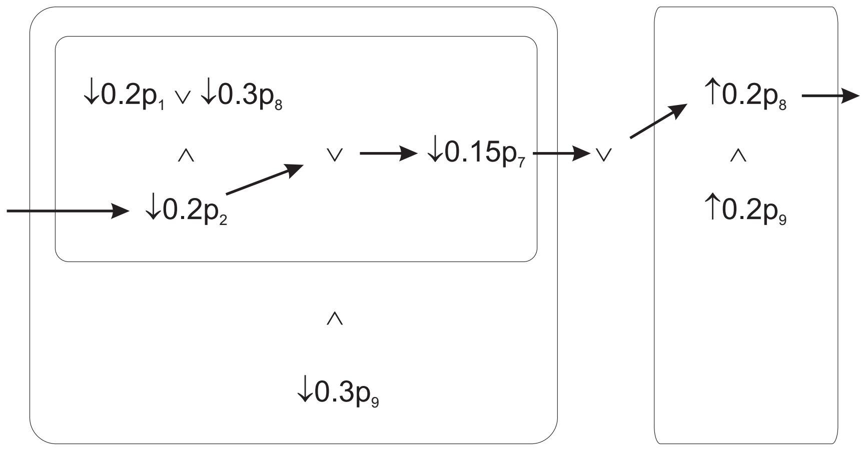


Figure 5.3: A path in the formula

5.5.2 Logical Calculus

The Logical Calculus for Regular HORN-NNF formulas is drawn by substituting the regular terms D^-, C^-, CNF^- and DNF^- in the previous four rules, by general regular NNF^- formulas. Also, the level of nesting allowed can be of any finite order.

As in the simple case, we need two kinds of Regular Unit Resolution rules:

- RDUR, for the case where the regular literal $(\downarrow j : p)$ of a unit regular clause $(\uparrow i : p)$ is in a disjunction, $i > j$ is true, and
- RCUR, similarly when $i > j$ is true and $(\downarrow j : p)$ is in a conjunction.

To extend the Logical Calculus of the Regular SIMPLE-HORN-NNF-SAT problem to the case of the general Regular HORN-NNF-SAT problem, we take separately each inference rule and we generalize the regular terms D^-, C^-, CNF^- and DNF^- existing in the inference rules by replacing them appropriately by general regular NNF^- terms.

Definition 5.5.3 *The rule RDUR for the Regular HORN-NNF formulas where the old terms have been substituted by general Regular NNF^- formulas is the following:*

$$\frac{(\uparrow i : p), (\{((\downarrow j : p) \vee NNF_1^-) \wedge NNF_2^-\} \vee NNF_3^- \vee C^+)}{(\{(NNF_1^-) \wedge NNF_2^-\} \vee NNF_3^- \vee C^+)}(DUR)$$

Now we need to extend the nesting level from 3 to any order k:

$$\frac{(\uparrow i : p), (\{(\ldots(\{((\downarrow j : p) \vee NNF_1^-) \wedge NNF_2^-\} \vee NNF_3^-) \wedge \ldots) \wedge NNF_{k-1}^-\} \vee NNF_k^- \vee C^+)}{(\{(\ldots(\{(NNF_1^-) \wedge NNF_2^-\} \vee NNF_3^-) \wedge \ldots) \wedge NNF_{k-1}^-\} \vee NNF_k^- \vee C^+)}(DUR)$$

Remark: Note that $i > j$ must be true to apply this inference rule. The regular term NNF_k can be a conjunct or a disjunct depending only on the number k of layers of nesting. If k is even, NNF_k is a regular conjunct and else, if k is odd, it is a regular disjunct.

Thus, the antecedents of the new inference rules are obtained from the previous ones by:

- substituting in the correspondent rule antecedent, the regular terms $(\downarrow j_2 : p_2 \vee \ldots \vee \downarrow j_n : p_n)$, $\{D_2^- \wedge \ldots \wedge D_k^-\}$ and DNF^- by general regular NNF^- formulas, and by
- generalizing from the level 3 of nesting of the operators $\vee/\wedge$ in Regular SIMPLE-HORN-NNF formulas to a general level k of nesting in HORN-NNF.

Then, with the antecedent defined, the consequent is easily derived. Applying the previous principle to the three inference rules of the previous LC for the Regular SIMPLE-HORN-NNF formulas, we obtain the definitive inference rules for the case of general Regular HORN-NNF formulas.

Definition 5.5.4 *Given that $i > j$ is true, the Logical Calculus for the case of general Regular HORN-NNF formulas is the following:*

$$\frac{(\{(\ldots\{((\{\Box \wedge NNF_1^-\} \vee NNF_2^-) \wedge NNF_3^-\} \vee \ldots \vee NNF_{k-1}^-) \wedge NNF_k^-\} \vee C^+)}{(\{(\ldots\{(NNF_2^-) \wedge NNF_3^-\} \vee \ldots \vee NNF_{k-1}^-) \wedge NNF_k^-\} \vee C^+)}(RSIMPLIF)$$

$$\frac{(\uparrow i : p), (\{(\ldots(\{(\downarrow j : p \vee NNF_1^-) \wedge NNF_2^-\} \vee NNF_3^-) \wedge \ldots) \wedge NNF_{k-1}^-\} \vee NNF_k^- \vee C^+)}{(\{(\ldots(\{(NNF_1^-) \wedge NNF_2^-\} \vee NNF_3^-) \wedge \ldots) \wedge NNF_{k-1}^-\} \vee NNF_k^- \vee C^+)}(RDUR)$$

$$\frac{(\uparrow i : p), (\{(\ldots\{(\{\downarrow j : p \wedge NNF_1^-\} \vee NNF_2^-) \wedge NNF_3^-\} \vee \ldots) \wedge NNF_{k-1}^-\} \vee NNF_k^- \vee C^+)}{(\{(\ldots\{(NNF_2^-) \wedge NNF_3^-\} \vee \ldots) \wedge NNF_{k-1}^-\} \vee NNF_k^- \vee C^+)}(RCUR)$$

$$\frac{(\{\uparrow i_1 : p_1 \wedge \ldots \wedge \uparrow i_k : p_k \wedge \ldots \wedge \uparrow i_n : p_n\})}{(\uparrow i_1 : p_1) \wedge \ldots \wedge (\uparrow i_k : p_k) \wedge \ldots \wedge (\uparrow i_n : p_n)}(RAE)$$

Now, we can state the first main theorem.

Theorem 5.5.1 *The previous Logical Calculus is Sound and Refutation Complete for the case of general Regular HORN-NNF formulas.*

Proof. The soundness is trivial and the completeness theorem proof is very similar to that of the logical calculus for the Regular SIMPLE-HORN-NNF-SAT problem. Let us take as an example the first lemma equivalent to lemma 5.4.3.

Lemma 5.5.1 *If* $\Gamma^+ \models (\uparrow i : p)$ *and* $rank(p) = k$ *then there exists at least one clause* $\mathcal{F} = (CNNF_1^- \vee CNNF_2^- \vee \ldots \vee CNNF_k^- \vee \{DNNF_1^- \wedge DNNF_2^- \wedge \ldots \wedge DNNF_n^-\} \vee \mathcal{C}^+)$ *such that*

1. $(\uparrow j : p) \in \mathcal{C}^+$*, or similarly,* $\mathcal{F} \in pos(p)$
2. $\forall(\downarrow j_i : p_i) \in \mathcal{F}$*,* $rank(p_i) < k$
3. $j > j_i$
4. $\forall CNNF_i^-$*,* $\exists DNNF^-$*,* $DNNF^- \in CNNF_i^-$ *s.t.* $\Gamma^+ \models \neg DNNF^-$ *and* $\exists DNNF_j^-$*,* $\forall CNNF^-$*,* $CNNF^- \in DNNF_j^-$ *s.t.* $\Gamma^+ \models \neg CNNF^-$*.*

The remaining lemmas of the proof are proved similarly using the same kind of generalization followed in the previous lemma with respect to the equivalent lemma in the MAIN THEOREM 1 for the Regular SIMPLE-HORN-NNF-SAT problem.

■

Thus, **MAIN THEOREM 1:** $\exists x, SRC(x)$ is true for the previous LC, namely we have, $SRC(LC)$.

5.5.3 A Quadratic Algorithm

The algorithmic version of the current Logical Calculus is the same that the previous one, defined in 5.4.23.

Definition 5.5.5 *The new inference rules are called Regular RAL, Regular RAC and Regular RACNF for* $RemoveAllRegularLiterals$*,* $RemoveAllRegularConjunctions$*,* $RemoveAllRegularCNFs$ *and they are defined as follows:*

$$\frac{(\uparrow i : p), \Gamma}{\Gamma = RemoveAllLiterals((\uparrow i : p), \Gamma)}(RegularRAL)$$

$$\frac{(\uparrow i : p), \Gamma}{\Gamma = RemoveAllConjunctions((\uparrow i : p), \Gamma)}(RegularRAC)$$

$$\frac{\Gamma}{\Gamma = RemoveAllCNFs(\Box, \Gamma))}(RegularRACNF)$$

As the calculus is the same that the previous one, the corresponding quadratic algorithm differs in only one point with respect to the previous one for the Regular SIMPLE-HORN-NNF-SAT problem.

In the current Regular HORN-NNF-SAT problem the nesting of formulas oblige to propagate the situations appearing when disjunctions are converted in the empty clause $\Box$. Removing terms in disjunctions can make reduce a disjunction to the empty clause. To its turn, this empty clause can be a term in a conjunction that contains the removed disjunction, and removing this conjunction could yield an empty disjunction. Thus, propagation of the empty clause in the formulas is required.

The following algorithm is a quadratic algorithm for the Regular HORN-NNF-SAT problem:

```
$SATALG_{Q1}(\Gamma)$
1  ApplyRAE($\Gamma$, Stack)
2  While $Stack \neq \{\}$ do:
3     $(\uparrow i : p) \leftarrow pop(Stack)$
4     RemoveAllLiterals($(\downarrow j : p), \Gamma$) when $i > j$
5     RemoveAllCNFs($\Box, \Gamma$)
6     RemoveAllConjunctions($(\downarrow j : p), \Gamma$) when $i > j$
7     Propagate($\Box$, $\Gamma$)
8     ApplyRAE($\Gamma$, Stack)
9  EndWhile
10 If $() = \mathcal{F} \in \Gamma$ then return("UNSAT") Else return("SAT")
```

Theorem 5.5.2 *The previous algorithm is quadratic.*

Proof. The quadratic complexity is derived from the quadratic complexity of the algorithm for the Regular SIMPLE-HORN-NNF-SAT problem. Indeed the function "Propagate" has the same complexity that the other operations in lines 4, 5, and 6, and as the maximal number of iterations is limited by the number of different propositions, the global complexity is quadratic. ■

This proves the MAIN THEOREM 3 for the Regular HORN-NNF-SAT problem:

$$CORSAT(SATALG_{Q1}) \wedge QSAT(SATALG_{Q1})$$

5.6 Conclusions

In this chapter, we have dealt with the Regular Simple-Horn-NNF SAT problem and the Regular Horn-NNF-SAT problem. The former is a special case of the latter.

For the first problem, we have proposed a Sound and Refutation Complete Logical Calculus and an almost linear SAT algorithm. Indeed, the non linear operation has been kept in the initialization procedure. The main deduction algorithm remains strictly linear.

For the Horn-NNF SAT problem, we have furnished a Sound and Refutation Complete Logical Calculus and a quadratic SAT algorithm.

On the theoretical part, these are (to our knowledge) the first tractability results concerning Regular non-clausal SAT problems. Indeed, until now, there are some well known results regarding Regular clausal satisfiability. Most of them have been referenced in this chapter. But, the Regular non-clausal tractability had not been tackled until now.

On the practical side, the algorithms developed can be of relevant interest to the Rule Based Systems. Actually, the multi-valued Regular Logic has been turned out to be an appropriate language to represent and reason with uncertainty. Many Expert Systems and other Rule Based Systems rely on this logic. Thus, with the developed algorithms the rules and questions in a Rule Based System can be extended allowing more general forms than the classical simple ones: a conjunction in the antecedent and a proposition in the consequent. Rules now can be composed by a Negation Normal Form term in the antecedent and a conjunction in the consequent.

The proposed algorithms run with the same complexity than their counterpart classical algorithms, but the proposed algorithms work with a smaller formula, and in the most favorable case, the difference in size of the classical conjunctions and the Negation Normal Form term could be of an exponential order. Thus, the gain in time, of the proposed method with respect to the classical algorithms can be of an exponential rate.

Chapter 6

Conclusions

In this thesis, we have dealt with the Satisfiability problem in Non-Clausal forms. More specifically, we have treated a kind of Non-Clausal Horn-like SAT problems in propositional logic and in multi-valued Regular Logic. For each logic, we have solved two main problems: The Simple-Horn-NNF SAT problem and the Horn-NNF SAT problem. The former is a particular case of the later.

The Simple-Horn-NNF SAT problem consists in verifying the satisfiability of a formula where the clauses $C = (CNF^- \vee DNF^- \vee C^+)$ have the following terms:

- A Conjunctive Normal Form term formed by a conjunction of disjunctions of negative literals $CNF^- = \{D_1^- \wedge D_2^- \wedge \ldots \wedge D_n^-\}$, where D_i^- in the propositional case is $D_i^- = (\neg p_{i,1} \vee \neg p_{i,2} \vee \ldots \vee \neg p_{i,n_i})$, and in the regular case is $D_i^- = (\downarrow i, i_1 : p_{i,1} \vee \downarrow i, i_2 : p_{i,2} \vee \ldots \vee \downarrow i, i_{n_i} : p_{i,n_i})$;
- A Disjunctive Normal Form term formed by a disjunction of conjunctions of negative literals $DNF^- = (C_1^- \vee C_2^- \vee \ldots \vee C_m^-)$, where C_j in the propositional case is $C_j = \{\neg p_{j,1} \wedge \neg p_{j,2} \wedge \ldots \wedge \neg p_{j,n_j}\}$, and in the regular case is $C_j^- = \{\downarrow j, i_1 : p_{j,1} \wedge \downarrow j, i_2 : p_{j,2} \wedge \ldots \wedge \downarrow j, i_{n_j} : p_{j,n_j}\}$;
- A conjunction formed by positive literals which in the propositional case is $C^+ = \{p_1 \wedge p_2 \wedge \ldots \wedge \ldots p_k\}$, and in the regular case is $C^+ = \{\uparrow i_1 : p_1 \wedge \uparrow i_2 : p_2 \wedge \ldots \wedge \uparrow i_n : p_n\}$.

In this thesis memory, we have provided:

- for the propositional SAT problem formed by a set of the described clauses, first a Sound and Refutation Complete Logical Calculus and second, a strictly linear SAT algorithm;
- for the Regular SAT problem formed by a set of the mentioned clauses containing regular literals -instead of propositional literals-, first a Sound and Refutation Complete Logical Calculus and second, an almost linear SAT algorithm (the non linear operations have been confined to the initialization procedure, the main algorithm is kept linear).

The more general Horn-NNF SAT problem has been studied also in this thesis, in the propositional case as well as in the Regular case. The Horn-NNF SAT problem consists in verifying the satisfiability of a set of clauses where each clause $C = NNF^{-} \vee C^{+}$ is formed by two terms:

- a Negation Normal Form term NNF^{-} formed exclusively by negative literals;
- a conjunction C^{+} formed exclusively by positive literals (equally to the previous simple case).

For the associated SAT problem composed by a set of theses clauses, we have provided:

- for the propositional case, a Sound and Refutation Complete Logical Calculus and a strictly linear SAT algorithm;
- for the Regular case, a Sound and Refutation Complete Logical Calculus and a quadratic SAT algorithm.

All the four proposed algorithms have been obtained by means of an original methodology that facilitates the proofs of correction and complexity of the issued algorithms. This methodology decomposes the proof of correction of a complex algorithm into the proof of correction of a Logical Calculus and a sequence of three algorithms. The last algorithm in the sequence is the final linear algorithm of complex structure.

Firstly, the proof of correction of the Logical Calculus follows standard mathematical technics. Afterwards, the proof of the correction of each of the three algorithms is based on the correction of its precedent algorithm in the sequence. This proof relies on the close similarity between two consecutive algorithms in the sequence.

On the practical side, the results obtained can be of benefic interest for the applications relying on the SAT problem such as Expert Systems, Hardware Design, Automated Software Verification, Symbolic Optimization, Logic Programming, Automated Theorem Proving, Truth Maintenance Systems, etc.

More specifically, the Rule Based Systems can take advantage of the proposed linear and almost linear algorithms developed in this thesis. Indeed, with the new algorithms the rules and questions in Rule Based Systems are allowed to have new more general forms. The rules now can be formed in its antecedent part by Negation Normal Forms (composed exclusively by positive -propositional and regular- literals) and in its consequent part by a conjunction of propositions or regular literals.

The interpreters of the Rule Based Systems constructed from these new algorithms introduce an exponential gain in time and in memory because they run in linear or almost linear time with exponentially less symbols than their counterpart classical interpreters.

On the theoretical side, we have pushed further the frontiers of tractability of the SAT problem. It has been established that new SAT problems can be resolved in linear and almost linear time. The first case applies to the non-clausal formulas in propositional logic and the second case to the non-clausal regular logic.

Bibliography

[1] G. Aguilera, I.P. de Guzman, and M. Ojeda. Increasing the efficiency of automated theorem proving. *Journal of Applied Non-classical Logics*, 5(1):9–29, 1995.

[2] E. Altamirano and G. Escalada-Imaz. Algoritmos óptimos para algunas teorías de Horn factorizadas. In *II Congrés Catalá d'Intel-ligència Artificial*, CCIA'99, pages 31–38, Girona, Spain, october 1999.

[3] E. Altamirano and G. Escalada-Imaz. Dos algoritmos eficientes para teorías de Horn factorizadas. In *V Conferencia de Ingeniería Eléctrica*, CIE'99, Mexico, D.F., september 1999.

[4] E. Altamirano and G. Escalada-Imaz. A quadratic algorithm for many-valued non-clausal Horn-like formulas. In *III Congrés Catalá d'Intel.ligència Artificial*, CCIA'00, pages 201–208, Vilanova i la Geltrú, Barcelona, Spain, october 2000.

[5] E. Altamirano and G. Escalada-Imaz. An almost linear class of multiple-valued non-clausal Horn formulas. In *X Congreso Español sobre Tecnologías y Lógica Fuzzy*, ESTYLF'00, pages 145–150, Sevilla, Spain, September 2000.

[6] E. Altamirano and G. Escalada-Imaz. An efficient proof method for non-clausal reasoning. In *XII International Symposium on Methodologies for Intelligent Systems*, volume 1932 of *LNAI*, pages 534–542, Charlotte, USA, October 2000. Springer-Verlag.

[7] E. Altamirano and G. Escalada-Imaz. Finding tractable formulas in NNF. In *I International Conference on Computational Logic*, volume 1861 of *LNAI*, pages 493–507, London, UK, July 2000. Springer-Verlag.

[8] E. Altamirano and G. Escalada-Imaz. Un Algoritmo casi lineal para fórmulas de Horn multivaluadas no clausales. In *VI Conferencia de Ingeniería Eléctrica*, CIE'00, Mexico, D.F., september 2000.

[9] E. Altamirano and G. Escalada-Imaz. Extending polynomiality to a class of non-clausal many-valued Horn-like formulas. In *6th European Conference ECSQARU 2001*, volume 2143 of *LNAI*, pages 792–804, Toulouse, France, September 2001. Springer-Verlag.

[10] P.B. Andrews. Theorem proving via general matings. *Journal of Association Computing Machinery*, 28, 1981.

[11] C. Ansótegui, R. Béjar, A. Cabiscol, and F. Manyà. The interface between P and NP in signed CNF formulas. In *Proceedings, 34th International Symposium on Multiple-Valued Logics (ISMVL), Toronto, Canada*, pages 251–256. IEEE CS Press, Los Alamitos, 2004.

[12] C. Ansótegui and F. Manyà. New logical and complexity results for Signed-SAT. In *Proceedings, 33rd International Symposium on Multiple-Valued Logics (ISMVL), Tokyo, Japan*, pages 181–187. IEEE CS Press, Los Alamitos, 2003.

[13] C. Ansótegui and F. Manyà. Mapping problems with finite-domain variables to problems with boolean variables. In *Proceedings of the 7th International Conference on the Theory and Applications of Satisfiability Testing, SAT-2004, Vancouver, Canada*, pages 111–119. Springer LNCS, 2004.

[14] B. Aspvall. Recognising disguised NR(1) instances of the satisfiability problem. *Journal of Algorithms*, (1):97–103, 1980.

[15] B. Aspvall, M.F. Plass, and R.E. Tarjan. A linear-time algorithm for testing the truth of certain quantified Boolean formulas. *Information Processing Letters*, 8(3):121–132, 1979.

[16] B. Beckert, R. Hähnle, and F. Manya. Transformations between signed and classical clause logic. In *Proc. Int. Symp. on Multiple Valued Logics, ISMVL'99*, Freiburg, Germany, 1999.

[17] W. Bibel. *Automated theorem proving.* Fiedr, Vieweg and Sohn, 1982.

[18] E. Boros, Y. Crama, P.L. Hammer, and M. Saks. A complexity index for satisfiability problems. *SIAM Journal on Computing*, (23):45–49, 1994.

[19] E. Boros, P.L. Hammer, and X. Sun. Recognition of q-Horn formulae in linear time. *Discrete Applied Mathematics*, (55):1–13, 1994.

[20] H.K. Buning, T. Lettmann, and C.J. van Rijsbergen. *Propositional Logic: Deduction and Algorithms.* Cambridge University Press, 1999.

[21] F. Bacchus C. Thiffault and T. Walsh. Solving Non-clausal Formulas with DPLL search. In *Tenth International Conference on Principles and Practice of Constraint Programming*, pages 663–678, 2004.

[22] V. Chandru and J.N. Hooker. Extended Horn sets in propositional logic. *Journal of ACM*, (38):205–221, 1991.

[23] M. Conforti, G. Cornuéjols, A. Kapoor, K. Vusković, and M.R. Rao. Balanced matrices. In J.R. Birge and K.G. Murty, editors, *Mathematical Programming: State of the Art.* 1994.

[24] S.A. Cook. The complexity of theorem-proving procedures. In *Third ACM Symposium on theory of Computing*, pages 151–158, 1971.

[25] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 4:394–397, 1962.

[26] T. Boy de la Tour. Minimising the number of clauses by renaming. In *CADE-10*, pages 558–572, 1990.

[27] A. del Val. On 2-SAT and Renamable Horn. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, pages 279–284, Austin, Texas, 2000.

[28] W.F. Dowling and J.H. Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *Journal of Logic Programming*, (3):267–284, 1984.

[29] H.B. Enderton. *A Mathematical Introduction to Logic, 2nd edition.* Academic Press, 2000.

[30] G. Escalada-Imaz. A quadratic algorithm and a linear algorithm for 2-CNF (in French). Technical Report LAAS-89378, Laboratoire D'Automatique et Analyse des Systemes, Toulouse, France, 1989.

[31] G. Escalada-Imaz. Moteurs d'Inférence Lineaires en Chaînage-Avant pour une classe de Systèmes de Règles. Technical Report LAAS-89172, Laboratoire D'Automatique et Analyse des Systemes, Toulouse, France, 1989.

[32] G. Escalada-Imaz. *Optimisation d'algorithmes d'inference monotone en logique des propositions et du premier ordre.* PhD thesis, Université Paul Sabatier, Toulouse, France, 1989.

[33] G. Escalada-Imaz and F. Manya. The satisfiability problem for multiple-valued horn formulae. In *Proc. International Symposium on Multiple-Valued Logics, ISMVL'94*, pages 250–256, Boston/MA, USA, 1994. IEEE Press, Los Alamitos.

[34] G. Escalada-Imaz and F. Manya. On the 2-SAT problem for signed formulas. In *Proc. Workshop/Conference on Many-Valued Logics for Computer Science Applications, COST Action 15.*, Barcelona, Spain, 1996.

[35] G. Escalada-Imaz and A.M. Martínez-Enríquez. Motores de Inferencia de Complejidad Optima de encadenamiento hacia adelante para diversas clases de sistemas de reglas. *Informática y Automática*, 27(3):23–30, 1994.

[36] S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM J. of Computing*, (5):691–703, 1976.

[37] J.W. Freeman. *Improvements to propositional satisfiability search algorithms.* PhD thesis, University of Pennsylvania, 1995.

[38] G. Gallo and G. Urbani. Algorithms for testing the satisfiability of propositional formulae. *Journal of Logic Programming*, (7):45–61, 1989.

[39] M.R. Genesereth and N.J. Nilsson. *Logical foundations of artificial intelligence.* Morgan Kaufmann, 1987.

[40] M. Ghallab and G. Escalada-Imaz. A linear control algorithm for a class of rule-based systems. *Journal of Logic Programming*, (11):117–132, 1991.

[41] M.T. Goodrich and R. Tamassia. *Data Structures and Algorithms in Java.* Jon Wiley and Sons, 2001.

[42] G. Gutiérrez, I. P. de Guzmán, J. Martínez, M. Ojeda-Aciego, and A. Valverde. Satisfiability testing for Boolean formulas using Δ-trees. *Studia Logica*, 72:33–60, 2002.

[43] R. Hähnle. Towards an efficient Tableau proof procedure for multiple-valued logics. In *Proc. of Computer Science Logic CSL'90, Heidelberg, Germany*, 1990.

[44] R. Hähnle. Uniform notation of Tableau rules for multiple-valued logics. In *Proc. ISMVL'91, Victoria, Canada*, 1991.

[45] R. Hähnle. A new translation from deduction into integer programming. In *Proc. Int. Conf. on Artificial Intelligence and Symbolic Mathematical Computing AISMC-1, Karlsruhe, Germany*, 1992.

[46] R. Hähnle. *Automated deduction in multiple-valued logics*, volume 10 of *International Series of Monographs in Computer Sciences.* Oxford University Press, 1993.

[47] R. Hähnle. Short conjunctive normal forms in finitely-valued logics. *Journal of Logic and Computation*, 4(6):905–927, 1994.

[48] R. Hähnle. Exploiting data dependencies in many-valued logics. *Journal of Applied Non-classical Logics*, (6):49–69, 1996.

[49] R. Hahnle and G. Escalada-Imaz. Deduction in many-valued logics: A survey. *Mathware and Soft Computing*, 4(2):69–97, 1997.

[50] R. Hähnle, N.V. Murray, and E. Rosenthal. Completeness for linear regular negation normal form inference systems. In *Proceedings ISMIS'97*, 1997.

[51] L. Henschen, E. Lusk, R. Overbeek, B.T. Smith, R. Veroff, S. Winker, and L. Wos. Challenge problem 1. *SIGART Newsletter*, (72):30–31, July 1980.

[52] L. Henschen and L. Wos. Unit refutations and Horn sets. *Journal of the Association for Computing Machinery*, 21(4):590–605, 1974.

[53] W. Hodges. Logical features of Horn clauses. In *Handbook of Logic in Artificial Intelligence and Logic Programming. Vol 1*, pages 459–466. 1993.

[54] J. Hsiang. Refutational theorem proving using term-rewriting systems. *Artificial Intelligence*, pages 255–300, 1985.

[55] P. Jackson and D. Sheridan. Clause Form Conversions for Boolean Circuits. In *7th International Symposium on the Theory and Applications of Satisfiability Testing*, pages 183–198, 2004.

[56] N. Jones and W. Laaser. Complete problems for deterministic polynomial time. *Theoretical Computer Science*, (3):105–117, 1977.

[57] R.M. Karp. Reducibility among combinatorial problems. In R.E. Miller and J.W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press. N.Y., 1972.

[58] S.C. Kleene. *Mathematical Logic.* Dover Publications, 2002.

[59] S.M. Leach, J.J. Lu, N.V. Murray, and E. Rosenthal. Mho-resolution: an inference rule for regular multiple-valued logics. In *6th European Workshop on Logics in AI JELIA'98*, 1998.

[60] H.R. Lewis. Renaming a set of clauses as a Horn set. *Journal of the ACM*, (25):134–135, 1978.

[61] J.J. Lu, N.V. Murray, and E. Rosenthal. Signed formulas and annotated logics. In *Proc. 23st International Symposium on Multiple-Valued Logic*, pages 48–53. IEEE Computer Society Press, Los Alamitos, 1993.

[62] J.J. Lu, N.V. Murray, and E. Rosenthal. A framework for automated reasoning in multiple-valued logics. *Journal of Automated Reasoning*, (21):39–67, 1998.

[63] F. Manya. *Proof Procedures for Multiple-Valued Propositional Logics.* PhD thesis, Universidad Autónoma de Barcelona, 1996.

[64] F. Manya. The 2-SAT problem in signed CNF formulas. *Journal of Multiple-Valued Logic*, 5(4):307–325, 2000.

[65] F. Manya, R. Béjar, and G. Escalada-Imaz. The satisfiability problem in regular cnf-formulas. *Soft Computing: A Fusion on Foundations, Methodologies and Applications*, 2(3):116–123, 1998.

[66] E. Mendelson. *Introduction to Mathematical Logic, Fourth Edition.* Chapman and Hall, 1997.

[67] M. Minoux. LTUR: A simplified linear-time unit resolution algorithm for Horn formulae and computer implementation. *Information Processing Letters*, (29):1–12, 1988.

[68] G. Mints. Gentzen-type systems and resolution rules, part 1: Propositional logic. In *Proc. COLOG-88, Tallin*, volume 417 of *LNCS*, pages 198–231. Springer, 1990.

[69] N.V. Murray. Completely Non-Clausal Theorem Proving. *Artificial Intelligence*, 18(1):67–85, 1982.

[70] N.V. Murray and E. Rosenthal. Dissolution: making paths vanish. *Journal of the ACM*, 3:504–535, 1993.

[71] J.A. Navarro and A. Voronkov. Generation of Hard Non-Clausal Random Satisfiability Problems. In *The Twentieth National Conference on Artificial Intelligence*, pages 436–436, 2005.

[72] N.J. Nilsson. *Principles of artificial intelligence.* Tioga Publishing Company, 1980.

[73] M. Ojeda-Aciego and A. Valverde. tascpl: TAS solver for Classical Propositional Logic. In *Logics in Artificial Intelligence, JELIA'04*, pages 731–735. Lect. Notes in Artificial Intelligence 3229, 2004.

[74] D.A. Plaisted. *Formal techniques in artificial intelligence*, chapter Mechanical Theorem Proving. Elsevier Science Publishers, 1990.

[75] R. Roy-Chowdhury-Dalal. Model theoretic semantics and tractable algorithm for CNF-BCP. In *Proc. of the AAAI-97*, pages 227–232, 1997.

[76] A. Sakharov. A transformational decision procedure for non-clausal propositional formulas. *CoRR*, cs.LO/0306035, 2003.

[77] J.S. Schlipf, F. Annextein, J. Franco, and R.P. Swaminathan. On finding solutions for extended Horn formulas. *Information Processing Letters*, (54):133–137, 1995.

[78] M.G. Scutellà. A note on Dowling and Gallier's top-down algorithm for propositional Horn satisfiability. *Journal of Logic Programming*, (8):265–273, 1990.

[79] R.M. Smullyan. *First-Order Logic.* Springer-Verlag, 1968.

[80] R. Socher. Optimising the clausal normal form transformation. *Journal of Automated Reasoning*, (7):325–336, 1991.

[81] Z. Stachniak. Non-clausal reasoning with propositional definite theories. In *International Conference on Artificial Intelligence and Symbolic Computation*, volume 1476 of *Lecture Notes in Computer Science*, pages 296–307. Springer Verlag, 1998.

[82] Z. Stachniak. Polarity guided tractable reasoning. In *International American Association on Artificial Intelligence, AAAI-99*, pages 751–758, 1999.

[83] Z. Stachniak. Going Non-clausal. In *Fifth International Symposium on the Theory and Applications of Satisfiability Testing*, pages 316–322, 2002.

[84] G. Tseitin. On the complexity of proofs in propositional logics. In J. Siekmann and G. Wrightson, editors, *Automation of Reasoning 2: Classical Papers on Computational Logic*, pages 466–483. Springer, 1983.