ISSN 2007-9737

Model Checking Algorithm for Repairing Inference between Conjunctive Forms

Guillermo De Ita, Pedro Bello

Benemérita Universidad Autónoma de Puebla,
Facultad de Ciencias de la Computación,
México

deitaluna63@gmail.com, pb5pbello@gmail.com

Abstract. Let K be a propositional formula and let ϕ be a query, the propositional inference problem $K \models \phi$ is a Co-NP-complete problem for propositional formulas without restrictions. We show the existence of polynomial-time cases for some fragments of propositional formulas K and ϕ that are different from the already known case of Horn formulas. For example, in the case that K is a formula in the fragment Krom or Horn or Monotone, then $K \models \phi$ can be decided in polynomial-time for any CNF ϕ.

Given two conjunctive normal forms (CNF's) K and ϕ, when $K \not\models \phi$, our proposal builds a minimal set of independent clauses S. The falsifying assignments of S are exactly the subset of models of K, which are not models for ϕ. In this way, the CNF S could extend the initial formula K in such a way that repair the inference, it is $(S \cup K) \models \phi$ is held.

Keywords. Propositional inference, repairing inference, reasoning s, NP-complete, Co-NP-complete.

1 Introduction

The primary goal of complexity theory is to classify computational problems according to their inherent computational complexity. A central issue in determining these frontiers has been the satisfiability problem (SAT) in propositional calculus.

The case 2-SAT, which consists in determining the satisfiability of propositions in two Conjunctive Normal Forms (2-CNF), is an important tractable case of SAT (see e.g. [2, 5, 12] for polynomial-time algorithms for 2-SAT). Meanwhile, if F is a 3-CNF formula, then the determination of the satisfiability of F is a classical NP-complete problem.

The analysis on the algorithms solving SAT problems has been helpful for delimiting frontiers between tractable and non-tractable problems.

In fact, the 2-CNF (Krom formulas) is a fragment of propositional formulas that has been very useful to clarify the computational time-complexity for different types of problems. Variations of the 2-SAT problem (i.e. in the optimization and counting area) have been key for establishing frontiers between tractable and intractable problems. Some problems related to 2-SAT remain intractable such as: Max-2SAT (finding an assignment that maximizes the number of satisfying clauses of F), Min-2SAT (finding a model of F with a minimum number of true variables) [12], and #2SAT (counting the number of models of F) [1, 6].

SAT problem has many applications and it has been shown to be fundamental for solving deductive reasoning problems [15, 17]. The automation of deductive reasoning is a basic and challenging logic problem [18]. Deductive propositional reasoning is usually abstracted as follows: Given a propositional formula K (capturing the knowledge about a domain), and a propositional formula ϕ (a query capturing the situation at hand), then the goal of reasoning is to determine whether K implies ϕ, which is presented as $K \models \phi$. This is known as the *propositional inference problem* (or the entailment problem).

We analyze here the computational complexity of the propositional inference problem $K \models \phi$, considering K and ϕ as conjunctive normal formulas (CNF).

We want to determine the characteristics on the formula K such that $K \models \phi$ is performed efficiently and determine the frontier when this inference problem changes to be intractable.

Since automatic reasoning is one of the purer forms of human intellectual thought, the automation of such reasoning by means of computers is a basic and challenging scientific problem [18]. Thus, one of the fundamental problems in automatic reasoning is the propositional inference problem. This last problem is a relevant task in many other issues such as: estimating the degree of belief, belief revision, abductive explanation, logical diagnosis, and many other procedures in Artificial Intelligence (AI) applications.

Inference operations are present not only in logic but in many areas of mathematics. The concept of logical inference has proven to be more fruitful for the development of a general logic theory than the concept of theorem and of logical validity. Abstract inference operations are known as closure operators in universal algebra and lattice theory [13]. In this research, we consider the inference problem $K \models \phi$ from an algorithmic point of view, instead of the algebraic perspective that Beyersdorff [3] used.

The current techniques that are most used to automatically check $K \models \phi$, when K and ϕ are CNF's, are the resolution method [4] and the systems type Gentzen [11]. However, these processes have not only an inherent exponential complexity, but also they lack of a recovery proposal when $K \not\models \phi$. On the contrary, our proposal for checking $K \models \phi$ allows to build a new CNF H for repairing $K \not\models \phi$ by $(K \cup H) \models \phi$. Notice that $K \cup H$ continues as a CNF; therefore, our proposal is closed on conjunctive forms.

Contribution In our main results, we show that apart from the fragment of Horn formulas, there are other fragments of the propositional calculus where $K \models \phi$ is polynomial-time decidable. We also present an algorithm for $K \models \phi$, when K and ϕ are two CNF's without restrictions. While $K \models \phi$ is been checked, our proposal also builds a minimal set of independent clauses S such that if $K \not\models \phi$, then S repairs the inference, that is, $(K \cup S) \models \phi$.

This is key for applications of automatic reasoning in belief revision and abductive explanation problems [7].

The remainder of this paper is organized as follows. In the following section some preliminaries are established. Section 3 introduces the main data structures to be used in this work. In this section also is analyzed the first simple cases of the propositional inference on limited fragments of conjunctive forms, and some efficient cases for the problem of inference between conjunctive forms are established. In section 4, the general problem of inference between conjunctive forms is analyzed, as well as the time-complexity analysis of our algorithmic proposal. Finally, in the last section the conclusions are presented.

2 Preliminaries

Let $X = \{x_1, \ldots, x_n\}$ be a set of n *Boolean variables*. A *literal* is either a variable x_i, or a negated variable $\overline{x}_i$. We indistinctly denote the negation of a literal l as $\overline{l}$ or $\neg l$. A variable $x \in X$ *appears* in a formula F only if either x or $\overline{x}$ is an element of F. The *size* of a CNF F, denoted as $|F|$, is defined as the sum of the occurrences of literals appearing in F.

A *clause* C is a disjunction of different and non-complementary literals. In this article, we discard the case of tautological clauses. For $k \in N$, a *k-clause* is a clause consisting of exactly k literals, and a $(\leq k)$-clause is a clause with at most k literals. A *phrase* D is a conjunction of literals, while a *k-phrase* is a phrase with exactly k literals.

A *conjunctive normal form* (CNF) F is a conjunction of non-tautological clauses. We say that F is a monotone positive CNF if all of its variables appear in unnegated form. An F is monotone when each literal of F appears with just one of its signs. When a literal l of F appears with only one of its signs, then l is called a pure literal in F.

A k-CNF is a CNF containing only *k-clauses*. $(\leq k)$-CNF denotes a CNF containing clauses with at most k literals. A 2-CNF formula F is said to be strict only if each clause of F consists exactly of two literals. In particular cases, we consider a CNF as a set of clauses.

A *disjunctive normal form* (DNF) is a disjunction of phrases, and a k-DNF is a DNF containing only k-*phrases*.

We use $v(X)$ to represent the variables appearing in X, where X can be a literal, a clause, a phrase, a DNF, or a CNF. For instance, for the clause $c = \{\overline{x}_1, x_2\}$, $v(c) = \{x_1, x_2\}$. In addition, we used $\neg Y$ as the negation operator of the logical element Y (a variable, a CNF, or a formula). $Lit(F)$ is the set of literals involved in F, even if they appear with only one of the two signs. For example, if $X = v(F)$, then $Lit(F) = X \cup \neg X = \{x_1, \overline{x}_1, ..., x_n, \overline{x}_n\}$. We denote $\{1, 2, ..., n\}$ by $[\![n]\!]$, and the cardinality of a set A by $|A|$.

An *assignment* s for a formula F is a function $s : v(F) \rightarrow \{0, 1\}$. An *assignment* s can also be considered as a set of literals without a complementary pair of literals, e.g., if $l \in s$, then $\bar{l} \notin s$. In other words, s turns l *true* and $\bar{l}$ *false*, or vice versa. Let C be a clause and s an assignment. Considering C as a set of literals, then we have that C is *satisfied* by s if and only if $(C \cap s) \neq \emptyset$. On the other hand, if for all $l \in C$, $\bar{l} \in s$, then s falsifies C. A phrase D is satisfied by an assignment s if $D \subseteq s$.

A *model* of F is an assignment for $v(F)$ that satisfies F. Meanwhile, a falsifying assignment of F is an assignment for $v(F)$ that contradicts F. Let F be a CNF, F is *satisfied* by an assignment s if each clause in F is satisfied by s. F is *contradicted* by s if any clause in F is falsified by s. Meanwhile, a DNF F is satisfied by s if any phrase in F is satisfied by s. F is contradicted by s if all phrases in F are contradicted by s.

If $n = |v(F)|$, then there are 2^n possible assignments defined over $v(F)$. Let $S(F)$ be the set of 2^n assignments defined over $v(F)$. $s \models F$ denotes that the assignment s is a model of F, and $s \not\models F$ denotes that s is a falsifying assignment of F. SAT(F) is the set containing all satisfying assignment of F (models of F), where specific values are assigned to all variables of F. On the other hand, FAL(F) is the set containing all falsifying assignments of F.

Considering a CNF F as a set of clauses, and $F_1 \subset F$ as a formula consisting of some (not all) clauses from F, if $v(F_1) \subset v(F)$, then an assignment over $v(F_1)$ is a *partial* assignment for F. If $n = |v(F)|$ and $n_1 = |v(F_1)|$, then any assignment over $v(F_1)$ has 2^{n-n_1} as assignment extensions over $v(F)$. If s has logical values determined for each variable in F, then s is a *total assignment*, or just an assignment of F.

We denote $F[s]$ as the operation of the substitution of the literals in F by the logical values taken for those literals in the assignment s, and after, to perform any tautological operation appearing in $F[s]$.

The notation SAT(F) is extended by considering general propositional formulas F, not only CNF's, but also when SAT(F) is the set of assignments defined on $v(F)$ such that $s \models F$. The SAT problem consists on determining if F has a model, i.e., if SAT$(F) \neq \emptyset$. 2-SAT denotes the SAT problem only defined on 2-CNF's. #SAT(F) denotes the counting problem of determining the number of models of F. #2SAT denotes #SAT for 2-CNF formulas. Meanwhile, #FAL(F) counts the number of falsifying assignments of F.

Clearly, for any propositional formula F, $S(F) = SAT(F) \cup FAL(F)$. Then, FAL$(F) = S(F) - SAT(F)$ by complementary properties on the set of assignments. On the other hand, if $n = |v(F)|$ then #FAL$(F) = 2^n$- #SAT(F).

It is known that the logical inference problem is a hard challenge in automatic reasoning, and it is a Co-NP-complete problem even in the propositional case. Let L be the set of all propositional formulas. $L(B)$ denotes the set of all propositional formula expressed in the fragment B of propositional calculus. For example, $L(DNF), L(Krom), L(Horn), L(Mon)$ denote the set of all Disjunctive Normal forms, 2-CNF's, Horn-formulas, and monotone formulas, respectively.

The problem to be analyzed here is the inference problem on fragments B of the propositional calculus, which is established as:

Problem IMP(B)
Instance: Let ϕ be a CNF and let K be a propositional formula expressed in the fragment B.

Question: Does $K \models \phi$ hold ?

We analyze the IMP(B) problem from an algorithmic point of view more than from an

algebraic perspective, as it was analyzed by Beyersdorff [3].

3 Inference on Fragments of Conjunctive Forms

Let F be a CNF and let l be a literal, the reduction of F by l, denoted by $F[l]$, is the formula generated by removing from F the clauses containing l (subsumption), and by removing $\bar{l}$ from the remaining clauses (unit resolution on F).

The reduction of a formula F by a set of literals can be inductively established as follows: let $s = \{l_1, l_2, \ldots, l_k\}$ be a partial assignment of $v(F)$. The reduction of F by s is defined by successively applying $F[l_i]$, l_i, $i = 1, \ldots, k$. The reduction of F by l_1 gives the formula $F[l_1]$, following a reduction of $F[l_1]$ by l_2, giving as a result the formula $F[l_1, l_2]$ and so on. The process continues until $F[s] = F[l_1, ..., l_k]$ is reached. In case that $s = \emptyset$ then $F[s] = F$.

Example 1. *Let* $F = \{\{x_1, \overline{x}_2\}, \{x_1, x_2\}, \{x_1, x_3\}, \{\overline{x}_1, x_3\}, \{\overline{x}_2, x_4\}, \{\overline{x}_2, \overline{x}_4\}, \{x_2, x_5\}, \{\overline{x}_5\}\}$. *Then,* $F[x_2] = \{\{x_1\}, \{x_1, x_3\}, \{\overline{x}_1, x_3\}, \{x_4\}, \{\overline{x}_4\}, \{\overline{x}_5\}\}$, *and If* $s = \{x_2, \overline{x}_3\}$, $F[s] = \{\{x_1\}, \{x_1\}, \{\overline{x}_1\}, \{x_4\}, \{\overline{x}_4\}, \{\overline{x}_5\}\}$.

Let K be a CNF and let s be a partial assignment of K. If a pair of contradictory unitary clauses is obtained while $K[s]$ is being computed, then K is falsified by the assignment s.

Furthermore, during the computation of $K[s]$, new unitary clauses can be generated. Thus, the partial assignment s is extended by adding the unitary clauses found, that is, $s = s \cup \{u\}$ where $\{u\}$ is a unitary clause. Moreover, $K[s]$ can be reduced again using the new unitary clauses. The above iterative process is generalized, and we call $Unit\_Propagation(K, s)$ to this iterative process. For simplicity, we abbreviate $Unit\_Propagation(K, s)$ as $UP(K, s)$.

As a result of applying $UP(K, s)$, we obtain a new assignment s' that extends to s, and a new subformula K' formed by the clauses from K that are not satisfied by s'. We denote as $K' = UP(K, s)$ to the formula K' resulting of the application of Unit Propagation on K by the assignment s.

Notice that if s falsifies K then s' could have complementary literals, and K' is unsatisfiable. In addition, when s satisfies K, then K' is empty.

The direct implementation of unit propagation takes time quadratic in the total size of the set to check, which is defined to be the sum of the size of all clauses, where the size of each clause is the number of literals it contains. Unit propagation can however be done in linear time by storing, for each variable, the list of clauses in which each literal is contained [19].

Let K be a CNF, $K = \bigwedge_{i=1}^{m} C_i$, where each C_i is a clause. For each clause C_i, $i \in [\![m]\!]$, the assignment s such that $C_i[s] = 1$, contains at least one literal of C_i, and if $C_i[s] = 0$, then all literal $l \in C_i$ appears as $\bar{l}$ in s. It is easy to build $FAL(K)$ when K is a CNF, since each clause $C_i \in K$ determines a subset of falsifying assignments of K. Therefore, $FAL(K)$ is the union of those falsifying assignments for each clause in K, this means that:

$$FAL(K) = \bigcup_{i=1}^{m} \{s \in S(K) \mid FAL(C_i) \subseteq s\}.$$

The following Lemma shows that the subsets of satisfiable formulas are also satisfiable.

Lemma 1. *If* K *is a satisfiable CNF, then* $\forall K' \subseteq K$, K' *is a satisfiable CNF.*

Proof. We know by the previous Lemma that $FAL(K) = \bigcup_{C_i \in K} FAL(C_i) \subset S(K)$, and this last containment is strict because K is satisfiable. Clearly, if we discard some clauses from K, forming K', then $FAL(K') = \bigcup_{C_i \in K'} FAL(C_i) \subseteq \bigcup_{C_i \in K} FAL(C_i) \subset S(K)$. Thus, K' is satisfiable. □

Lemma 2. *Let* K *be an unsatisfiable CNF,* $\forall$ *CNF* K' *such that* $K \subseteq K'$, K' *remains unsatisfiable.*

Proof. For an unsatisfiable CNF K, it holds that $FAL(K) = \bigcup_{C_i \in K} FAL(C_i) = S(K)$. Therefore, if we add new clauses to K forming K', then $FAL(K) = \bigcup_{C_i \in K} FAL(C_i) \subseteq \bigcup_{C_i \in K'} FAL(C_i)$. Since $\bigcup_{C_i \in K} FAL(C_i) = S(K)$, then we obtain that also $\bigcup_{C_i \in K'} FAL(C_i) = S(K)$. Thus, K' is unsatisfiable. □

Let K and ϕ be two propositional formulas, we say that K implies ϕ, written as $K \models \phi$, if ϕ is satisfied for each model of K, i.e., if $SAT(K) \subseteq SAT(\phi)$. The last containment can be expressed via the falsifying sets of the formulas as in the following Lemma:

Lemma 3. *$FAL(\phi) \subseteq FAL(K)$ if and only if $K \models \phi$.*

Proof. This property comes from the basic properties on sets that are closed under complementation and because of $S(K) = SAT(K) \cup FAL(K)$. Then, $(FAL(\phi) \subseteq FAL(K)) \equiv (SAT(K) \subseteq SAT(\phi)) \equiv K \models \phi$. □

3.1 Building Strings to Falsify Clauses and to Satisfy Phrases

Let K and ϕ be two CNF's, $K = \bigwedge_{i=1}^{m} C_i$ and $\phi = \bigwedge_{i=1}^{k} \varphi_i$. Each set $FAL(C_i)$ can be represented in a succinct way via a string A_i of length $n = |\upsilon(K)|$. Given a clause $C_i = (x_{i_1} \vee \ldots \vee x_{i_k})$, then the value at each position from i_1-th to i_k-th of the string A_i is fixed with the truth value that falsifies each literal of C_i. E.g., if $x_{i_j} \in C_i$, the i_j-th element of A_i is set to 0. On the other hand, if $\overline{x}_{i_j} \in C_i$, then the i_j-th element is set to 1.

The variables in $\upsilon(K)$ which do not appear in C_i are represented by the symbol *, meaning that they could take any logical value in the set $\{0, 1\}$. In this way, the string A_i of length $n = |\upsilon(K)|$ represents the set of assignments falsifying the clause C_i. E.g. if $K = \{C_1, \ldots, C_m\}$ is a 2-CNF, $n = |\upsilon(K)|$, $C_1 = \{x_1 \vee x_2\}$ and $C_2 = \{x_2 \vee \overline{x}_3\}$, the assignments of $FAL(C_1)$ can be represented by the string $00 * * \ldots *$ and the assignments of $FAL(C_2)$ are represented by the string $*01 * \ldots *$.

We call *falsifying string* to the string A_i that represents the set of falsifying assignments of a clause C_i. We denote by $Fals(C_i)$, the string (with n symbols), that is the *falsifying string* of the clause C_i. For purposes of evaluating a falsifying string $s = Fals(C_i) = s_1 s_2 \ldots s_k$ on a formula F, we would consider s as only one assignment on the set of variables $X = \{x_1, \ldots, x_k\}$ of F. The evaluation $F[s]$ is defined in orded to iterate on each symbol $s_i, i \in [\![k]\!]$ of the string, this is, $F[s] = F[s_1, \ldots, s_k]$, in the following way. $F[s_i] = F$, if $s_i = *$. $F[s_i] = F[x_i]$ when $s_i = 1$, and $F[s_i] = F[\overline{x}_i]$ for $s_i = 0$.

On the other hand, let $\sigma = (l_1 \wedge \ldots \wedge l_j)$ be a phrase defined over $Lit(K)$. A string v_σ of n symbols is associated with σ, and each one of its values $v_\sigma[i], i = 1, \ldots, n$ is determined as:

$$v_\sigma[i] = \begin{cases} 1 & if\ x_i \in \sigma, \\ 0 & if \overline{x}_i \in \sigma, \\ * & if\ neither\ x_i \notin \sigma\ nor\ \overline{x}_i \notin \sigma. \end{cases} \quad (1)$$

The string v_σ is a succinct form to represent SAT(σ), because any assignment s over K is a satisfying assignment for σ, when s and v_σ coincide their values (0 or 1) in the same positions,

Thus, v_σ represents the set of $2^{n-|\sigma|}$ satisfying assignments for σ. Therefore, we call v_σ as the *satisfying string* of the phrase σ.

The use of strings to represent a set of assignments has been very useful to design reasoning procedures, e.g. for computing belief revision operators, or not redundant families of subcubes [10, 16].

3.2 Basic Cases for the Inference Problem $K \models \phi$

Let $X = \upsilon(K)$ and $Lit(K) = X \cup \overline{X}$ be the set of variables and the set of literals appearing in K, respectively. Let us assume K a satisfiable formula, and ϕ that consists of only one literal l. For this class of formulas, we analyze what happens to $K \models l$.

— Case $l \notin Lit(K)$: When $l \notin \upsilon(K)$ then $K \not\models l$, because any model m of K can be extended as $m \cup \{\neg l\}$ that is a satisfying assignment for K, but falsifies l.

— Case $l \in Lit(K)$: Consider the formula $K_1 = K[\neg l]$. In this case, if SAT$(K_1) \neq \emptyset$ then $K \not\models l$, because any model m of K_1 can be extended as $m \cup \{\neg l\}$ that is a model for K but falsifies l. Otherwise, SAT$(K_1) = \emptyset$ and $K \models l$, because there are not models for K doing false to l.

Now, let us consider the case where ϕ is formed by just one clause, $\phi = (l_1 \vee \ldots \vee l_k)$. An initial situation is given by $\upsilon(K) \cap \upsilon(\phi) = \emptyset$.

Theorem 1. *Let K be a satisfiable CNF and let ϕ be a non-tautology clause. If $(\upsilon(K) \cap \upsilon(\phi)) = \emptyset$, then $K \not\models \phi$.*

Proof. Let $s \in Fals(\phi)$, s exists since ϕ is a non-tautology, and additionally, s has not assigned values for variables in K, since $(\upsilon(K) \cap \upsilon(\varphi)) = \emptyset$. Let $u \in Sat(K)$, u exists since K is satisfiable. Let $v = s \cup u$ be an assignment for all variables in K and ϕ. v is a valid assignment because it has non contradictory literals, since $(\upsilon(K) \cap \upsilon(\phi)) = \emptyset$. Furthermore, $\phi[v] = false$ and $K[v] = true$ by construction of v, and then, $K \not\models \phi$. Thus, v is a witness of the succinct disqualification of $K \models \phi$. □

On the other hand, let us assume that ϕ continues as one clause $\phi = (l_1 \vee \ldots \vee l_k)$, and that $\upsilon(K) \cap \upsilon(\phi) \neq \emptyset$. In this case, let $K_1 = UP(K, [\bar{l}_1, \ldots, \bar{l}_k])$. If K_1 is satisfiable, then $K \not\models \phi$; otherwise $K \models \phi$. When K_1 is satisfiable, then a model m for K_1 exists and the assignment $s = m \cup \{\bar{l}_1, \ldots \bar{l}_k\}$ is valid since it does not have complementary literals, and s holds $K[s] = true$ and $\phi[s] = false$. Thus, $K \not\models \phi$. Notice that the computational complexity for determining $K \not\models \phi$ is inherently related of the computational complexity of determining Sat(K_1).

Lemma 4. *Let $s \in Fals(\phi)$. If $K' = UP(K, s)$ is formed by clauses with at least one pure literal, then $K \not\models \phi$.*

Proof. Let $s \in Fals(\phi)$ and let $K' = UP(K, s) = C_1 \vee \ldots \vee C_k$. Let $u_1 = \bigcup_i l_i$, where l_i is a pure literal in $C_i \in K'$, then $K'[u_1] = true$ since each clause in K' has at least one pure literal. Let $u = u_1 \cup s$ be a valid assignment, since $\upsilon(K') \cap \upsilon(\phi) = \emptyset$, and then $K \not\models \phi$, by theorem 1 . □

Let $K = K_1 \cup K_2$, where for each $C \in K_2$ there is a literal $l_i \in C$ that is pure in K_2 and $l_i \notin Lit(\phi)$.

Lemma 5. *$K \models \phi$ if and only if $K_1 \models \phi$.*

Proof. ($\Leftarrow$) If $K_1 \models \phi$ then $FAL(\phi) \subseteq FAL(K_1)$. Also, $FAL(K_1) \subseteq FAL(K)$ since K contains a K_1, and it is possible that K has more clauses than those in K_1. By property of subsets $FAL(\phi) \subseteq FAL(K)$, and then $K \models \phi$, by Lemma 3.

($\Rightarrow$) By contradiction. Assume that $K \models \phi$ and $K_1 \not\models \phi$. If $K_1 \not\models \phi$, then an assignment $s \in Fals(\phi)$ exists, and $K_1[s] = true$. Let $u_1 = \bigcup_i l_i$, where l_i is a pure literal in K_2. Thus, $K_2[u_1] = true$. Let $v = s \cup u_1$, v is a valid assignment because any l_i is pure in K_2 ($l_i \in u_1$) and $\bar{l}_i \notin s$ (since $l_i \notin Lit(\phi)$). Furthermore, $K[v] = true$ and $\phi[v] = false$, and then $K \not\models \phi$ - a contradiction. Therefore, $K_1 \models \phi$. □

Let us consider the problem of propositional inference: $K \models \phi$, where K and ϕ are two general CNF's. The decision problem $K \models \phi$ is a Co-NP-complete problem for CNF's in general [3], because $K \models \phi$ is equivalent to proving that $(\neg K \vee \phi)$ is a tautology. But, $(\neg K)$ is a DNF, due to the D'Morgan law, that is distributed on a CNF ϕ, and then $(\neg K \vee \phi)$ generates a DNF. And the tautology problem on a DNF is a classic Co-NP-complete problem. In particular, a formula K' in 3-DNF can be written as $K \models \phi$ with $\phi = false$ and $K = \neg K'$ a 3-CF. Then, any algorithm for deciding $K \models \phi$ also determine the tautologicity of K', therefore $K \models \phi$ is a Co-NP-complete even if K is a 3-CNF.

The fragment of propositional logic in which $K \models \phi$ is known to have a polynomial-time decidable method is when both K and ϕ are Horn formulas. For this case, there exist linear-time procedures for deciding $K \models \phi$ [8, 11]. However, adding 2-CNF formulas as extensions of Horn formulas would change the inference procedure into an exponential-time complexity process on the number of Horn inferences to perform.

For example, let $(K \wedge H)$ be an input formula, where K is a Horn formula and H is a monotone 2-CNF, and let ϕ be a Horn formula. If we want to decide $(K \wedge H) \models \phi$, then we could apply the distributive property on each monotone positive binary clause $(x \vee y) \in H$ and the Horn part K. Therefore, $K \wedge (x \vee y) \models \phi$ if and only if $(\neg(K \wedge (x \vee y)) \vee \phi)$ is valid, and it holds if and only if $((\neg K \vee (\neg x \wedge \neg y)) \vee \phi) \equiv (((\neg K \vee \neg x) \wedge (\neg K \vee \neg y)) \vee \phi) \equiv ((\neg(K \wedge x) \wedge \neg(K \wedge y)) \vee \phi) \equiv (\neg(K \wedge x) \vee \phi) \wedge (\neg(K \wedge y) \vee \phi) \equiv ((K \wedge x) \models \phi) \wedge ((K \wedge y) \models \phi)$.

Thus, for each positive monotone binary clause, we duplicate the number of Horn inferences to perform. If we consider the existence of two monotone binary clauses in H, that is $(K \wedge (x_1, y_1) \wedge$

$(x_2, y_2)) \models \phi$, and we apply the distribution on literals of the monotone clauses, then we obtain four Horn inferences: $((K \wedge x_1 \wedge x_2) \models \phi)$, $((K \wedge x_1 \wedge y_2) \models \phi)$, $((K \wedge y_1 \wedge x_2) \models \phi)$ and $((K \wedge y_1 \wedge y_2) \models \phi)$. If there are m positive monotone binary clauses $(x_i \vee y_i), i = 1, \ldots, m$ in H, we would have under the above reduction, a total of 2^m Horn inferences, type: $(H \wedge (x_1 \vee y_1) \wedge \ldots \wedge (x_m \vee y_m)) \models \phi$. which leads to an exponential-time complexity process on the number of Horn inferences to perform.

This exponential growth on the Horn inference, when positive monotone clauses are been considered, has been one of the limitations in the development of disjunctive logic programming software. One of the researching lines for tackling the time-complexity of non-Horn logic programming has been to compile the knowledge base and new information into more simple forms, for example, using implicant prime forms [14].

The principle of resolution has been a practical method to check unsatisfiability between conjunctive forms. However, the inference based on resolution method has intrinsic limitations [4].

Despite of the methods of refutation commonly used in the Horn inference, here we consider other kind of methods to determine whether $K \models \phi$. For the fragment of CNF's, our method focuses on checking that $FAL(\phi) \subseteq FAL(K)$ in order to prove $K \models \phi$.

Let us consider from now on that $\upsilon(\phi) \subseteq \upsilon(K)$. Let $l \in Lit(K)$. We determine the sets: K_l, $K_{\neg l}$, and K_0 formed by the clauses from K containing l, $\bar{l}$ and neither l nor $\bar{l}$, respectively. We define also $K_{-(l)} = \{D : (D \vee l) \in K_l\}$. Let $K' = K_{-(l)} \cup K_0$. Notice that in this case, $K' = K[\bar{l}]$. If K' is satisfiable, then $K \not\models l$, because any model m of K can be extended as $m \cup \{\bar{l}\}$ that is a model for K that falsifies l. Otherwise, $K \models l$, because all falsifying assignment for l also falsifies K.

Lemma 6. *Let K be a formula in the fragment of Krom or Horn or Monotone, and let s be a partial assignment defined on $\upsilon(K)$, then $K[s]$ continues as a Krom, or Horn or Monotone formula, respectively.*

Proof. Note that for the fragment considered in this lemma, K can also be seen as a CNF, $K' = \bigwedge_{i=1}^{m} C_i$. Let s be a partial assignment of K. Let $K' = \bigwedge_{i=1}^{m} C_i[s]$, the resulting conjunctive form from $K[s]$. $C[s]$ does not increment the length $|C|$ neither change the parity of literals.

Thus, if K is in the fragment of Krom or Horn or Monotone, then K' continues in the same logic fragment of K. □

The following theorem shows that the computational complexity to determine $K \models \phi$, when K is in the fragment of Krom or Horn or Monotone is related to the computational complexity of $SAT(K)$.

Theorem 2. *Let ϕ be a CNF and let K be a formula in the fragment Krom or Horn or Monotone. For this fragment of propositional formulas, $K \models \phi$ is decided in polynomial-time.*

Proof. We present as proof, a polynomial-time algorithm.

For each $\varphi_i \in \phi$
{ Let $s_i = Fals(\varphi_i)$. Let $K_i = UP(K, s_i)$.
If $(SAT(K_i))$ then $K \not\models \phi$ /*Because any model m of K_i can be extended as $m \cup s_i$ that is a model for K but it falsifies ϕ */.
}
Returns($K \models \phi$) /*Because $FAL(\phi) \subseteq FAL(K)$ */

The loop consists of at most $|\phi|$ iterations, proving in each step: $SAT(K_i)$.

This is decided in polynomial-time since $SAT(K_i)$ is in the complexity class P for formulas K_i in the fragment Krom or Horn or Monotone formulas.

Thus, the total time complexity of the procedure is polynomial. □

From the previous theorem, we delimit the frontier between tractable and intractable instances for $K \models \phi$, being K and ϕ CNF's. We have shown that $K \models \phi$ is solved in polynomial-time when K is a 1-CNF or a 2-CNF. Meanwhile, $K \models \phi$ is a Co-NP-complete problem even if K is a 3-CNF.

ISSN 2007-9737

4 An Exact Algorithm for $K \models \phi$, when K and ϕ are CNF's

We will present in this section a general method for checking $K \models \phi$, when K and ϕ are two CNF's. Since K and ϕ are CNF's, then $FAL(\phi)$ and $FAL(K)$ can be denoted through the falsifying strings of their clauses. When $K \not\models \phi$, then $FAL(\phi) \not\subset FAL(K)$.

In this case, $FAL(\phi) - FAL(K) \neq \emptyset$ and this implies the existence of a set of assignments S in which $S \subseteq FAL(\phi)$ and $S \not\subset FAL(K)$. Then, an algorithm for checking $K \models \phi$ could consists in computing $FAL(\phi) - FAL(K)$. Assuming $K = \bigwedge_{i=1}^{m} C_i, \phi = \bigwedge_{i=1}^{k} \varphi_i$, we want to determine:

$$\forall i \in [\![k]\!] : \left(S_i = FAL(\varphi_i) - \bigcup_{j=1}^{m} FAL(C_j) \right). \quad (2)$$

We have designed a procedure to compute $FAL(\varphi_i) - FAL(K)$ based on the application of the *independence property* introduced by Dubois [9].

Definition 1. *Given two clauses C_i and C_j, if they have at least one complementary literal, it is said that C_i and C_j are* independent. *Otherwise, we say that both clauses are* dependent.

Definition 1 can be written in terms of falsifying strings as follows:

Given two falsifying strings $A = Fals(C_i)$ and $B = Fals(C_j)$ each one of length n, if there is an $i \in [\![n]\!]$ such that $A[i] = x$ and $B[i] = 1 - x$, $x \in \{0, 1\}$, then the clauses C_i, C_j (as well as its falsifying strings) have the *independence* property. Otherwise, we say that both clauses (strings) are *dependent*.

This means that the falsifying strings for independent clauses have complementary values (0 and 1) in at least one of their fixed values.

Let $K = \{C_1, C_2, \cdots, C_m\}$ be a CNF of m clauses. K is called independent if each pair of clauses $C_i, C_j \in K, i \neq j$ has the independence property. Otherwise, K is called dependent.

Let $F = \{C_1, C_2, \cdots, C_m\}$ be a CNF with m clauses defined on $n = |\upsilon(F)|$ variables. Let $C_i, i \in [\![m]\!]$ be a clause in F such that $|C_i| < n$, and $x \in (\upsilon(F) - \upsilon(C_i))$ be any variable. We have that:

$$C_i \equiv (C_i \vee \overline{x}) \wedge (C_i \vee x). \quad (3)$$

Definition 2. *Given a pair of dependent clauses C_1 and C_2, if $Lit(C_1) \subseteq Lit(C_2)$, then we say that C_2 is subsumed by C_1.*

If C_1 subsumes C_2, then $FAL(C_2) \subseteq FAL(C_1)$. On the other hand, if C_2 is not subsumed by C_1 and they are dependent, there is a set of indices $I = \{1, \ldots, p\} \subseteq \{1, \ldots, n\}$ such that for each $i \in I, x_i \in C_1$, but $x_i \notin C_2$. There exists a reduction to make C_2 independent from C_1. We call this transformation the *independent reduction* between two clauses, and it works as follows. Let C_1 and C_2 be two dependent clauses.

Let $\{x_1, x_2, \ldots, x_p\} = Lit(C_1) \setminus Lit(C_2)$. By (3) we can write: $C_1 \wedge C_2 \equiv C_1 \wedge (C_2 \vee \neg x_1) \wedge (C_2 \vee x_1)$. Now C_1 and $(C_2 \vee \neg x_1)$ are independent. Applying (3) to $(C_2 \vee x_1)$:

$$C_1 \wedge C_2 \equiv$$
$$C_1 \wedge (C_2 \vee \neg x_1) \wedge (C_2 \vee x_1 \vee \neg x_2) \wedge (C_2 \vee x_1 \vee x_2).$$

The first three clauses are independent. If we repeat the above process of making the last clause independent from the previous clauses until x_p is achieved, then $(C_1 \wedge C_2)$ is equivalent to the following set of independent clauses: $C_1 \wedge (C_2 \vee \neg x_1) \wedge (C_2 \vee x_1 \vee \neg x_2) \wedge \ldots \wedge (C_2 \vee x_1 \vee x_2 \vee ... \vee \neg x_p) \wedge (C_2 \vee x_1 \vee x_2 \vee ... \vee x_p)$.

The last clause contains all literals of C_1, so it is subsumed by C_1, and then:

$$\begin{aligned} C_1 \wedge C_2 \quad &\equiv \quad C_1 \wedge (C_2 \vee \neg x_1) \wedge (C_2 \vee x_1 \vee \neg x_2) \wedge \\ \ldots \quad &\wedge \quad (C_2 \vee x_1 \vee x_2 \vee ... \vee \neg x_p). \end{aligned} \quad (4)$$

We obtain on the right side of (4) an independent set of $p + 1$ clauses. We denote this independence reduction as $ind\_reduct(C_1, C_2)$. We will use the independent reduction between two clauses C and φ (or between their respective falsifying strings) in order to define:

$$Ind(C, \varphi) = \begin{cases} \varphi & \text{If } \varphi \text{ and } C \text{ are independent,} \\ \emptyset & \text{If } \varphi \text{ is subsumed by } C, \\ ind\_reduct(C, \varphi) - C & \text{Otherwise.} \end{cases} \quad (5)$$

The following theorem shows that the independent operator (5) builds a set of clauses, which covers exactly the space allocations that conforms $FAL(\varphi_i) - FAL(C_j)$.

Theorem 3. *Let φ and C be two clauses, then $FAL(Ind(C,\varphi)) = FAL(\varphi) - FAL(C)$.*

Proof. If $Ind(C,\varphi) = \emptyset$, then $FAL(\varphi) \subseteq FAL(C)$. Therefore, $FAL(\varphi) - FAL(C) = \emptyset$. Now, we assume that $Ind(C,\varphi) \neq \emptyset$. Let s be an assignment such that $s \in FAL(Ind(C,\varphi))$. We will show that $s \in FAL(\varphi)$ and $s \notin FAL(C)$. If $s \in FAL(Ind(C,\varphi))$, then s falsifies φ because each clause in $Ind(C,\varphi)$ has the form $(\varphi \vee R)$ where R is a disjunction (R could be empty, for example in the case $Ind(c,\varphi) = \emptyset$). If s falsifies $(\varphi \vee R)$, then s has to falsify φ, and thus $s \in FAL(\varphi)$.

Algorithm 1 Procedure $Ind(K,\phi)$

```
Input: K: A KB, φ: a CNF
V = ∅ {Temporal stack for processing each φ_i ∈ φ}
for all φ_i ∈ φ do
  Push(φ_i, V); Fs = ∅; {Output in Fs a CF (Set of clauses)}
  for all C_j ∈ K do
    while (V ≠ ∅) do
      φ = Pop(V); {Try following clause}
      Fs = Fs − φ; {Remove the exit clause}
      Nc = Ind(φ, C_j); {Form: Nc ∧ C_j ⊨ φ }
      if (Nc ≠ ∅) then
        Fs = Fs ∪ Nc; {Only if there are clauses to add}
      end if
    end while
    for all φ ∈ Fs do
      Push(φ, V); {Next iteration considers new clauses}
    end for
  end for
  S_i = Fs; {S_i = Ind(K, φ_i)}
end for
```

On the other hand, each clause $(\varphi \vee R) \in Ind(C,\varphi)$ is independent from C because of the construction of the operator Ind; therefore, $FAL(C) \cap FAL(Ind(C,\varphi)) = \emptyset$. Thus, $s \notin FAL(C)$. □

Theorem 4. $(C \cup Ind(C,\varphi)) \models \varphi$.

Proof. If φ and C are independent, then $Ind(C,\varphi) = \varphi$. Therefore, $(C \wedge Ind(C,\varphi)) \equiv (C \wedge \varphi) \models \varphi$ by the propositional property $(p \wedge q) \supset q$ and by reflexivity $q \models q$. Otherwise, we assume $(C \wedge Ind(C,\varphi)) \equiv (C \wedge \varphi)$ by Equation 4, and by the property $(C \wedge \varphi) \models \varphi$, then the theorem holds. □

Let φ be a clause, and $K = \bigwedge_{j=1}^{m} C_j$. If we apply the Ind operator between each C_j and φ, we get as result S_i. The union of each S_i, $S = \cup_{i=1}^{m} S_i$ holds that $S \subseteq FAL(\varphi)$ and $S \not\subset FAL(K)$. The pseudo-code for computing $Ind(K,\phi)$ is shown in the Algorithm 1.

In order to generate a minimum set of independent clauses as a result of $Ind(K,\varphi)$, it is crucial to sort the clauses $C_j \in K$ in ascending order according to the length $|S_j| = |Lit(C_j) - Lit(\varphi)|$. This is done since the number of literals in C_j, different to the literals in φ, determine the number of independent clauses to be generated. Hence, we have a strategy for reducing the number of independent clauses to be generated by each $Ind(C_j,\varphi), \forall C_j \in K$.

The operator Ind applied on the clause φ, and on each one of the clauses $C_j \in K$, allows us to build the space $FAL(\varphi) - FAL(K)$. Thus, the following recurrence is defined as: $A_1 = \varphi$, $A_{j+1} = Ind(C_j, A_j)$.

In order to perform $Ind(C_j, A_j)$, the remaining clauses in $C_l \in K, l = j+1, \ldots, m$ (those that are not reduced independently with A_j) are sorted again in ascending order according to the number of common literals between the literals represented by A_j.

The above process can be extended to each $\varphi_i \in \phi, i = 1, \ldots, k$, as follows:

$$A_{i,1} = \varphi_i.$$

$A_{i,j+1} = Ind(C_j, A_{i,j}), j = 1, \ldots, m$ and $i = 1 \ldots k$.

The clauses $A_{i,m+1}$ comply with $\bigcup_{i=1}^{k}(A_{i,m+1}) = FAL(\phi) - FAL(K)$. These strings $A_{i,j}, i = 1, \ldots, k, j = 1, \ldots, m$ form a matrix of strings, as it is illustrated in Table 1. Notice that if $A_{i,j} = \emptyset$, then $A_{i,l} = \emptyset$, for $l = j+1, \ldots, m$.

Example 2. *Let us illustrate our procedure on* K *a 2-CNF and a general CNF* ϕ*. Let* $K = \{(x_1, x_2), (x_1, x_7), (\overline{x}_1, x_7), (\overline{x}_2, x_3), (x_3, \overline{x}_4), (x_5, \overline{x}_6), (x_6, x_7)\}$ *and* $\phi = \varphi_1 \wedge \varphi_2 \wedge \varphi_3 = \{(\overline{x}_3, x_6), (x_2, \overline{x}_6, x_7), (x_1, x_4, x_5)\}$*.*

In each cell of the Table 1, the result of $Ind(C_j, \varphi_i)$ *is shown until we determine* $K \models \varphi_i, i = 1, \ldots, 3$*. Although* $K \models \varphi_2$*, the procedure reports that* $K \not\models \phi$*, because of* $K \not\models \varphi_1$ *and* $K \not\models \varphi_3$*. However, our procedure also builds a set of independent clauses whose falsifying assignments cover exactly the set of models of* K*, which are not models for* φ_1 *and* φ_3*. This is shown in the last column of Table 1.*

As a result of Table 1, let $H = Ind(K, \phi) = \{(\overline{x}_1, \overline{x}_3, x_6, \overline{x}_7), (x_1, \overline{x}_2, \overline{x}_3, x_6, \overline{x}_7), (x_1, \overline{x}_2, \overline{x}_3, x_4, x_5, x_6, \overline{x}_7)\}$*. Notice that* $K \cup H$ *continues being a CNF. The new CNF* H *holds* $FAL(H) \subseteq FAL(\phi)$*. Furthermore,* $FAL(H)$ *represents the minimum set of models for* K *that are not models for* ϕ*, since* $FAL(H) = FAL(\phi) - FAL(K)$*. Therefore, we have that* $(K \cup H) \models \phi$ *by Theorem 4. Although* K *and* ϕ *are 2-CNF's,* $H = Ind(K, \phi)$ *could not be a 2-CNF, which means that our procedure is not closed under 2-CNF's.*

4.1 Time-Complexity Analysis

Given K and ϕ two conjunctive normal forms, the time-complexity of our method that checks $K \models \phi$ depends on the maximum number of clauses that can be generated through $Ind(K, \varphi_i)$ for each $\varphi_i \in \phi$. $Ind(K, \varphi_i)$ generates the empty set in some cases (when $K \models \varphi_i$). However, in the worst case, the time complexity for calculating $Ind(K, \varphi_i)$ depends on the length of the sets: $S_{ij} = \{x_1, x_2, \ldots, x_p\} = Lit(C_j) - Lit(\varphi_i), j = 1, \ldots, m$.

In order to simplify our analysis, let us consider as first case when K is a strict 3-CNF. For this case, $K \models \phi$ is an intractable problem. As it was previously mentioned, given $\varphi_i \in \phi$, the sets $S_{ij} = Lit(C_j) - Lit(\varphi_i), j = 1, \ldots, m$ are sorted in ascending order on the number of literals. The case $|S_{ij}| = 0$ indicates that C_j and φ have the independence property, or that φ_i is subsumed by C_j. In any case, no new strings are formed by $Ind(C_j, \varphi_i)$.

$Fals(\varphi_i)$ arranges logical values for a set of variables that do not change value during the process $Ind(K, \varphi_i)$; therefore, $\mid S_{i1} \mid + \mid S_{i2} \mid + \ldots + \mid S_{im} \mid \leq n - \mid \varphi_i \mid$. And, in the worst case, $S_{ij} = \{x_1, x_2\} = Lit(C_j) - Lit(\varphi_i)$ has two literals, because the case where $|S_{ij}| = 3$ already determines that $K \not\models \varphi_i$ (by Theorem 1).

Therefore, $Ind(C_j, \varphi_i)$ generates at most two new clauses C_a and C_b holding $C_a = (\varphi_i \vee \neg x_1)$ and $C_b = (\varphi_i \vee x_1 \vee \neg x_2)$. This conforms a Fibonacci sequence on the length of clauses formed by $Ind(C_j, \varphi_i)$. Considering $x = |\varphi_i|$, then the number of clauses generated by $Ind(K, \varphi_i)$ is modeled by the Fibonacci recurrence: $T(x) = T(x+1) + T(x+2)$.

It is known that the growth of the Fibonacci sequence $T(n)$ is upper bounded by Φ^{n-1}, with $\Phi = \frac{1+\sqrt{5}}{2}$ - known as the "golden ratio".

Let us consider $Poly(n, m)$ as a polynomial function that represents the computational time spent in sorting the clauses in S_{ij}, to review for subsumed clauses and to perform the independence reduction between two clauses.

Thus, when K is a 3-CNF, the time complexity in the worst case for checking $K \models \varphi_i$ has an upper bound of $O(\Phi^{n-|\varphi_i|} * Poly(n, m))$, where Φ is the "golden ratio", and $n = |v(K)|$.

If $K \not\models \phi$, then $Ind(K, \phi)$ allows to build a new CNF whose falsifying assignments cover $FAL(\phi) - FAL(K)$ exactly. Consequently, in this way the initial inference is repaired by $(K \cup Ind(\phi, K)) \models \phi$. Repairing the initial inference problem is relevant for different problems in automatic reasoning. For example, in propositional belief revision [7], as well as other automatic reasoning problems between conjunctive forms.

5 Conclusion

The inference problem is key to improve automatic reasoning methods. We show the existence of polynomial-time inference methods for some fragments (apart from the Horn fragment) of propositional logic. For example, we have shown that if K is expressed via a 2-CNF and ϕ is a CNF without restrictions, then the

Table 1. Computing $Ind(K, \phi)$

$FAL(\varphi_1)$				$FAL(K)$			
	01*	*10****	****01*	*****00	1*****0	0*****0	00*****
*1**0*	**1**0*	**1**0*	**1**01	**1**01	**1**01	**1**01	1*1**01
							011**01
$FAL(\varphi_2)$				$FAL(K)$			
	*****00	*00****	0*****0	1*****0	****01*	*10****	**01***
0***10	*0***10	10***10	10***10	$K \models \varphi_2$			
$FAL(\varphi_3)$				$FAL(K)$			
	****01*	**01***	1*****0	00*****	0*****0	*10****	*****00
0**00**	0**000*	0**000*	0**000*	01*000*	01*0001	0110001	0110001

propositional inference problem $K \models \phi$ is solved in polynomial-time.

Furthermore, given two CNF's K and ϕ, our proposal to check $K \not\models \phi$ builds a set of independent clauses S such that the falsifying assignments of S are exactly the subset of models of K, which are not models for ϕ.

As a result, this builds the set S of necessary clauses that repair the inference problem, this is $(S \cup K) \models \phi$. This is key in applications of automatic reasoning on propositional formulas.

References

1. **Arad, I., Santha, M., Sundaram, A., Zhang, S. (2019).** Linear-time algorithm for quantum 2sat. Theory of Computing, Vol. 14(1), pp. 1–27.
2. **Aspval, B., Plass, M., Tarjan, R. (1979).** A linear-time algorithm for testing the truth of certain quantified boolean formulas. Information Processing Letters, Vol. 8(3), pp. 121–123.
3. **Beyersdorff, O., Meier, A., Thomas, M., Vollmer, H. (2009).** The complexity of propositional implication. Information Processing Letters, Vol. 109(18), pp. 1071–1077.
4. **Bordeaux, L., Hamadi, Y., Zhang, L. (2006).** Propositional satisfiability and constraint programming: A comparative survey. ACM Computing Surveys, Vol. 38, pp. 1–54.
5. **Buresh-Oppenheim, J., Mitchell, D. (2007).** Minimum 2cnf resolution refutations in polynomial time. Proc. SAT'07 - 10th int. Conf. on Theory and applications of satisfiability testing, LNCS, pp. 300–313.
6. **Dahllöf, V., Jonsson, P., Wahlström, M. (2005).** Counting models for 2sat and 3sat formulae. Theoretical Computer Science, Vol. 332(1-3), pp. 265–291.
7. **De-Ita, G., Marcial, R., Bello, P., Contreras, M. (2018).** Belief revisions between conjunctive normal forms. Journal of Intelligent & Fuzzy Systems, Vol. 34, pp. 3155–3164".
8. **Dowling, W., Gallier, J. (1984).** Linear-time algorithms for testing the satisfiability of propositional horn formulae. Journal of Logic Programming, Vol. 1(3), pp. 267–284.
9. **Dubois, O. (1991).** Counting the number of solutions for instances of satisfiability. Theoretical Computer Science, Vol. 81, pp. 49–64.
10. **Ellis, D. (2010).** Irredundant families of subcubes. Mathematical Proc. of the Cambridge Philosophical Society, Cornell University, pp. 1–24.
11. **Gallier, J. (2003).** Logic for Computer Science, chapter Foundations of Automatic Theorem Proving (chapter 9). University of Pennsylvania, Department of Computer and Information Science, pp. 410–447.
12. **Gusfield, D., Pitt, L. (1992).** A bounded approximation for the minimum cost 2-sat problem. Algorithmica, Vol. 8, pp. 103–117.

13. **Jansana, R. (2011).** Propositional Consequence Relations and Algebraic Logic. Edward N. Zalta (ed.).
14. **Marchi, J., Bittencourt, G., Perrusssel, L. (2010).** Prime forms and minimal change in propositional belief bases. Ann. Math. Artif. Intelligence, Vol. 59, pp. 1–45.
15. **Moshman, D. (2004).** From inference to reasoning: The construction of rationality. Thinking & Reasoning, Vol. 10(2), pp. 221–239.
16. **Pozos-Parra, M., Weiru, L., Perrussel, L. (2013).** Dalal's revision without hamming distance. Lecture Notes in Artificial Intelligence, Vol. 8265, pp. 41–53.
17. **Roth, D. (1996).** On the hardness of approximate reasoning. Artificial Intelligence, Vol. 82, pp. 273–302.
18. **Shankar, N. (1997).** Cambridge Tracks in Theoretical Computer Science No. 38. Cambridge University Press.
19. **Zhang, H., Stickel, M. (1996).** An efficient algorithm for unit-propagation. Fourth Int. Symposium on Artificial Intelligence and Mathematics, Fort Lauderdale, pp. 166–169.

Article received on 24/07/2021; accepted on 15/09/2021.
Corresponding author is Guillermo De Ita.