The Incremental Satisfiability Problem for a Two Conjunctive Normal Form

Guillermo De Ita[1], J. Raymundo Marcial-Romero[2] and J. A. Hernández[2]

[1] Faculty of Computer Sciences, BUAP
[2] Facultad de Ingeniería, UAEM
deita@cs.buap.mx, jrmarcialr@uaemex.mx, xoseahernandez@gmail.com.mx

Abstract. We propose a novel method to review $K \vdash \phi$ when K and ϕ are both in Conjunctive Normal Forms (CF). We extend our method to solve the incremental satisfiablity problem (ISAT), and we present different cases where ISAT can be solved in polynomial time.
Keyword: Satisfiability Problem, Incremental Satisfiability Problem, 2-SAT, Entail Propositional Problem, Efficient Satisfiability Instances.

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 the propositional calculus. The case 2-SAT, to determine the satisfiability of propositional two Conjunctive Normal Forms (2-CF), is an important tractable case of SAT. Variations of the 2-SAT problem, e.g. in the optimization and counting area, have been key for establishing frontiers between tractable and intractable problems.

One of the fundamental problems in automatic reasoning is the propositional entail problem which is a relevant task in many other issues such as estimating the degree of belief, to review or update beliefs, the abductive explanation, logical diagnosis, and many other procedures in Artificial Intelligence (AI) applications.

It is known that logic entail problem is a hard challenge in automatic reasoning due to it is co-NP-Hard even in the propositional case [8]. However, some fragments of propositional logic allow efficient reasoning methods [3]. One of the most relevant cases of efficient reasoning is the fragment of Horn Formulas. We present a novel method to solve the entail problem between conjunctive forms and how to apply it for solving the incremental satisfiability problem (ISAT).

Hooker [7] presented an algorithm for the ISAT problem, in which the main contribution was the speed-up for solving a single formula by solving a growing subset of its constraints. Whittemore et al. [11] defined the incremental satisfiability problem as the solving of each formula in a finite sequence of subformulas. Wieringa [12] presented an incremental satisfiability solver and some of its applications. Finally, Nadel [10] presented a variation of ISAT problem under assumptions that are modeled as first decision variables; all inferred clauses that depend on some of the assumptions include their negation.

We present here an algorithm for solving the 2-ISAT problem, and we establish an upper bound for the time-complexity of 2-ISAT, as well as, we show some efficient cases for the ISAT and the 2-ISAT problems.

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 $\overline{x}_i$. As usual, for each $x \in X$, $x^0 = \overline{x}$ and $x^1 = x$.

A *clause* is a disjunction of different and non-complementary literals. Notice that we discard the case of tautological clauses. For $k \in I\!N$, 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* is a conjunction of literals, a *k-phrase* is a phrase with exactly k literals.

A *conjunctive normal form* (CNF, or CF) F is a conjunction of clauses. We say that F is a monotone positive CF if all of its variables appear in unnegated form. A k-CF is a CF containing only *k-clauses*. $(\leq k)$-CF denotes a CF containing clauses with at most k literals. A 2-CF formula F is said to be strict only if each clause of F consists of two literals. A *disjunctive normal form* (DF) is a disjunction of phrases, and a k-DF is a DF containing only *k-phrases*.

A variable $x \in X$ *appears* in a formula F if either x or $\overline{x}$ is an element of F. We use $\upsilon(X)$ to represent the variables involved in the object X; where X can be a literal, a clause, or a CF. $Lit(F)$ is the set of literals involved in F, i.e. if $X = \upsilon(F)$, then $Lit(F) = X \cup \overline{X} = \{x_1, \overline{x}_1, ..., x_n, \overline{x}_n\}$. Also we used $\neg Y$ as the negation operator on the object Y. 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 $s : \upsilon(F) \to \{0, 1\}$. 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 $\overline{l} \notin s$. Let c be a clause and s an assignment, c is *satisfied* by s if and only if $c \cap s \neq \emptyset$. On the other hand, if for all $l \in c$, $\overline{l} \in s$, then s falsifies c.

Let F be a CF, 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. A *model* of F is an assignment for $\upsilon(F)$ that satisfies F. A falsifying assignment of F is an assignment for $\upsilon(F)$ that contradicts F. A DF F is satisfied by s if any phrase is satisfied by s. F is contradicted by s if all of its phrases are contradicted by s.

If $n = |\upsilon(F)|$, then there are 2^n possible assignments defined over $\upsilon(F)$. Let $S(F)$ be the set of 2^n assignments defined over $\upsilon(F)$. $s \vdash F$ denotes that the assignment s is a model of F. $s \not\vdash F$ denotes that s is a falsifying assignment of F. If $F_1 \subset F$ is a formula consisting of some clauses from F, and $\upsilon(F_1) \subset \upsilon(F)$, an assignment over $\upsilon(F_1)$ is a *partial* assignment over $\upsilon(F)$. If s has logical values determined for each variable in F then s is a *total assignment* of F.

The SAT problem consists on determining whether F has a model. SAT(F) denotes the set of models of F, then SAT$(F) \subseteq S(F)$. The set FAL$(F) = S(F) \setminus SAT(F)$ consists of the assignments from $S(F)$ that falsify F. Clearly, for any propositional formula F, $S(F) = SAT(F) \cup FAL(F)$.

A Knowledge Base (KB) is a set K of formulas. Given a KB K and a propositional formula ϕ, we say that K implies ϕ, and we write $K \vdash \phi$, if ϕ is satisfied for each model of K, i.e., if $SAT(K) \subseteq SAT(\phi)$. This last problem is known as the *propositional entail* problem. The incremental satisfiability problem (ISAT) consists in deciding if an initial knowledge Base K keeps its satisfiability anytime a conjunction of new clauses ϕ is added.

3 Computing falsifying assignments of CF's

Assume a KB K in CF, $K = \bigwedge_{i=1}^{m} C_i$, where each C_i is a clause, $i \in [\![m]\!]$. For each clause C_i, $i \in [\![m]\!]$, the assignment $s(C_i) = 1$ contains at least one literal in C_i. It is easy to build $FAL(K)$ since each clause $C_i \in K$ determines a subset of falsifying assignments of K. The following lemma expresses how to form the falsifying set of assignments of a CF.

Lemma 1 *Given a CF* $K = \bigwedge_{i=1}^{m} C_i$*, it holds that* $FAL(K) = \bigcup_{i=1}^{m} \{\sigma \in S(K) \mid FAL(C_i) \subseteq \sigma\}$

Lemma 2 *If a CF K is satisfiable, then $\forall K' \subseteq K$, K' is a satisfiable CF.*

Corollary 1 *If a CF K is unsatisfiable, then $\forall$ CF K' such that $K \subseteq K'$, K' remains unsatisfiable.*

Now, let us consider the propositional entail problem: $K \vdash \phi$, where K and ϕ are CF's. The decision problem $K \vdash \phi$ is a classical Co-NP-Complete problem for CF's in general, since this problem is logically equivalent to the tautology problem for any DF, which is a classic co-NP-complete problem.

On the other hand, $K \vdash \phi$ iff $SAT(K) \subseteq SAT(\phi)$, and this last goal is equivalent to prove $FAL(\phi) \subseteq FAL(K)$, due to basic properties on sets that are closed under complementation.

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

The best known case of an efficient method for the inference $K \vdash \phi$ between CF's is when both K and ϕ are Horn formulas. In this case, the application of SLD-resolution leads to a linear-time process for deciding $K \vdash \phi$. The application of SLD-resolution has been the mechanism most commonly used in the development of logic programming languages [5].

However, including 2-CF's as extensions of a Horn formula and continue applying SLD-resolution method as the inference engine, gives an exponential-time complexity process on the number of Horn inferences to perform.

Despite of the refutation methods commonly used in the Horn inference, we consider here another method to determine whether $K \vdash \phi$. When $K = \bigwedge_{i=1}^{m} C_i$ and $\phi = \bigwedge_{i=1}^{k} \varphi_i$, our method focuses on checking that $FAL(\phi) \subseteq FAL(K)$ in order to prove $K \vdash \phi$.

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 $C_i = (x_{i_1} \vee \ldots \vee x_{i_k})$, the value at each position from i_1-th to i_k-th of the string A_i is fixed with the truth value falsifying 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.

We call *falsifying string* to the string A_i representing the set of falsifying assignments of a clause C_i. We denote by $Fal\_String(C_i)$, the string (with n symbols), that is the *falsifying string* for the clause C_i. As K and ϕ are CF's, the falsifying strings of their clauses allow us to denote $FAL(\phi)$ and $FAL(K)$. If $K \nvdash \phi \equiv FAL(\phi) \not\subseteq FAL(K)$ implies that there exists a set of assignments S such that $S \subseteq FAL(\phi)$ and $S \not\subseteq FAL(K)$. A reviewing procedure for $K \vdash \phi$ consists on taking each falsifying string representing $FAL(\phi)$ and reviewing if it is a subset of $FAL(K)$, e.g:

$$\forall i \in [[k]] : \Big(FAL(\varphi_i) \subseteq \bigcup_{j=1}^{m} FAL(C_j)\Big) \text{ where } K = \bigwedge_{i=1}^{m} C_i, \phi = \bigwedge_{i=1}^{k} \varphi_i$$

4 An exact algorithm for $K \vdash \phi$, when K and ϕ are CF's

We present a method for checking $K \vdash \phi$, with K and ϕ CF's.

Definition 1 *Given two clauses C_1 and C_2, if they have at least one complementary literal, it is said that they have the* independence *property. Otherwise, we say that the clauses are* dependent.

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

Let $F = \{C_1, C_2, \cdots, C_m\}$ be a CF, $n = |\upsilon(F)|$. Let $C_i, i \in [\![m]\!]$ be a clause in F and $x \in \upsilon(F) \setminus \upsilon(C_i)$ be any variable, we have that $C_i \equiv (C_i \vee \neg x) \wedge (C_i \vee x)$

Definition 2 *Given a pair of dependent clauses C_1 and C_2, if $Lit(C_1) \subseteq Lit(C_2)$ 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 dependents, 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 transform C_2 to become independent from C_1, we call this transformation as the *independent reduction* between two clauses that works as follows: let C_1 and C_2 be two dependent clauses. Let $\{x_1, x_2, \ldots, x_p\} = Lit(C_1) \setminus Lit(C_2)$. Thus, $C_1 \wedge C_2 \equiv C_1 \wedge (C_2 \vee \neg x_1) \wedge (C_2 \vee x_1)$. Now C_1 and $(C_2 \vee \neg x_1)$ are independent. Then, $C_1 \wedge C_2 \equiv C_1 \wedge (C_2 \vee \neg x_1) \wedge (C_2 \vee x_1 \vee \neg x_2) \wedge (C_2 \vee x_1 \vee x_2)$

The first three clauses are independent. Repeating the process of making the last clause independent with the previous ones, until x_p is considered; we have that $C_1 \wedge C_2$ can be written as:
$C_1 \wedge (C_2 \vee \neg x_1) \wedge (C_2 \vee x_1 \vee \neg x_2) \wedge \ldots \wedge (C_2 \vee x_1 \vee x_2 \vee \ldots \vee \neg x_p) \wedge (C_2 \vee x_1 \vee x_2 \vee \ldots \vee x_p)$.

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

$$C_1 \wedge C_2 \equiv C_1 \wedge (C_2 \vee \neg x_1) \wedge (C_2 \vee x_1 \vee \neg x_2) \wedge \ldots \wedge (C_2 \vee x_1 \vee x_2 \vee ... \vee \neg x_p) \quad (1)$$

We obtain on the right hand side of (1) an independent set of $p+1$ clauses which we denote as $indep\_reduction(C_1, C_2)$. We use the independent reduction between two clauses C and φ to define:

$$Ind(C, \varphi) = \begin{cases} \varphi & \text{if } \varphi \text{ and C are independent} \\ \emptyset & \text{if } \mathrm{Lit}(C) \setminus \mathrm{Lit}(\varphi) = \emptyset \\ indep\_reduction(C, \varphi) - C & \text{Otherwise} \end{cases}$$

The operation $Ind(C, \varphi)$ forms a conjunction of clauses whose falsifying assignments are exactly $FAL(\varphi) - FAL(C)$.

Theorem 1 *If φ and C are two clauses, then $FAL(Ind(C, \varphi)) = FAL(\varphi) - FAL(C)$*

Let $K = \bigwedge_{j=1}^{m} C_j$ be a CF and φ be a clause. If we apply the Ind operator between each $C_j \in K$ and φ, we get as a result a set S such that $S \subseteq FAL(\varphi)$ and $S \not\subset FAL(K)$.

In order to generate a minimum set of independent clauses as a result of $Ind(K, \varphi)$, it is crucial to sort the clauses $C_j \in K$ according to the length $|S_j| = |Lit(C_j) \setminus Lit(\varphi)|$ in ascending order, because the number of literals in C_j, different to the literals in φ, determines the number of independent clauses to be generated.

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

In order to perform $Ind(C_j, A_j)$, the remaining clauses in $C_l \in K, l = j+1, \ldots, m$, those which are not reduced independently with A_j, are sorted again in ascendent order according to the number of common literals with the literals represented by A_j. This process can be extended to each $\varphi_i \in \phi, i = 1, \ldots, k$, as: $A_{i,1} = \varphi_i; A_{i,j+1} = Ind(C_j, A_{i,j}), j = 1, \ldots, m$, and $i = 1, \ldots, k$ being so constructed clauses $A_{i,m+1}$ such that $\bigcup_{i=1}^{k}(A_{i,m+1}) = FAL(\phi) - FAL(K)$. These strings $A_{i,j}, i = 1, \ldots, k, j = 1, \ldots, m$ form a matrix of strings. Notice that if $A_{i,j} = \emptyset$ then $A_{i,l} = \emptyset$, for $l = j+1, \ldots, m$.

5 Incremental Satisfiability Problem

Definition 3 *Let F be a 2-CF and L its set of literals. The relation $\rightarrow_R \subset L \times L$ is defined as follows: $x \rightarrow_R y$ if and only if $x \rightarrow y$.*

The transitive closure of $\rightarrow_R$, denoted by "$\Rightarrow$" allows to define $T(x) = \{x\} \cup \{y : x \Rightarrow y\}$.

Definition 4 *Let F be a 2-CF, for any literal $x \in F$, it is said that $T(x)$ is inconsistent if $\overline{x} \in T(x)$ or $\perp \in T(x)$, otherwise $T(x)$ is said to be consistent.*

Definition 5 *A base for the set of models of a 2-CF F, denoted as $S(F)$, is a partial assignment s of F which consists of the variables with a fixed truth value.*

The incremental satisfiability problem (ISAT) involves checking whether satisfiability is maintained when new clauses are added to an initial satisfiable knowledge base K. ISAT is considered as a generalization of SAT since it allows changes of the input formula over the time, and also, it can be considered as a prototypical Dynamic Constraint Satisfaction Problem (DCSP) [9].

Different methods have been applied to solve ISAT, among them, branch and bounds procedures as variants of the classical Davis-Putnam-Loveland (DPL) method, denoted as IDPL methods. In IDPL procedures, when adding new clauses, those procedures maintain the search tree generated previously for the set of clauses K. IDPL performs substantially faster than DPL for a large set of SAT problems [7].

As a generalization of SAT, ISAT has been considered as an NP Problem, although until now, we have not seen complexity theory studies about the complexity-time differences between SAT and ISAT. For example, it is known that 2-SAT is in the complexity class P, however it is not known the computational complexity of 2-ISAT. It is clear that a set of changes over a satisfiable KB K in 2-CF could change K into a general CF, in whose case, it turns in a general CF K', $K \subset K'$, and where the SAT problem on K' is a classic NP-complete problem.

Rather than solving related formulas separately, modern solvers attempt to solve them *incrementally* since many practical applications require solving a sequence of related SAT formulas [2, 4]. We present in this section, a study about the threshold for the 2-ISAT problem that could be relevant to understand the border between P and NP complexity classes.

Assuming an initial KB K, and a new CF ϕ to be added, both are satisfiable CF's, let us consider some cases where ISAT can be determined directly.

1. If K and ϕ are 2-CF's then $(K \wedge \phi)$ is a 2-CF that is the input of ISAT, and in this case, 2-ISAT is solvable in linear-time by applying the well known algorithms for 2-SAT [6, 1]
2. If ϕ consists of one clause and we have the satisfiability tree of K, we only have to review which satisfiable branches of the tree falsify ϕ, and this can be done in linear time on the number of satisfiable branches of the tree.
3. For monotone formulas, ISAT keeps satisfiable formulas. If each variable maintains an unique sign in both K and ϕ then $(K \wedge \phi)$ is always satisfiable.

Let us consider now that K is a 2-CF and ϕ is a general CF, both of them different from the previous cases. By the results presented in Section 4, each $\varphi_i \in \phi$ such that $K \vdash \varphi_i$ is removed from ϕ, so we assume that $\phi = (\phi - \varphi_i)$. It means, we will consider only the clauses in ϕ which decrease effectively the set of models of K. Assume that the computation of both $T(x)$ and $B_i = FAL(\varphi_i)$ have been computed for each $x \in Lit(K)$ and each $\varphi_i \in \phi$, respectively.

The proposal for reviewing the satisfactibility of $(K \wedge \phi)$ is based on the following properties:

1. Given the partial assignments A_1, A_2 which they are part of any model (if there exists) of K. Those partial assignments may be extended in a way that they do not falsify any $\varphi_i \in \phi$, which is verified by $Ind(A_j, \varphi_i), j = 1, 2$. If it is possible, then a model for $(K \wedge \phi)$ is built.
2. Otherwise, $Ind(A_j, \varphi_i) = \emptyset, j = 1, 2$ and any model of K will be part of any falsifying assignment of ϕ. Thus, $(K \wedge \phi)$ is unsatisfiable.

When the operation $Ind(A_1, \varphi_i)$ is applied, a new literal x will be joined to A_1. In fact, we consider to join $T(x)$ instead of x. For example, if $A_1 = *01{*}{*}{*}1*$ and $\varphi_1 = *0*010**$, $Ind(A_1, \varphi_i)$ gives as a result $*011**1*$, $*0100*1*$ and $*010101*$. However, in this case it means to build the following three parcial assignments: $A_1 \cup T(x_4)$, $A_1 \cup T(\neg x_4) \cup T(\neg x_5)$, and $A_1 \cup T(\neg x_4) \cup T(x_5) \cup T(\neg x_6)$. If any of those strings is inconsistent then such string is substituted by $\emptyset$.

Let us analyze the growth on the number of possible partial assignments of the operation: $Ind(A_1, \varphi_i)$, $\forall \varphi_i \in \phi$. Firstly, the number of partial assignments for a fixed φ_i is $|S| = |v(\varphi_i) - v(A_1)|$. Moreover, each partial assignment $s_i \in S$ is independent to the other assignments in S, because they are different in at least one literal, and each of them hold $|s_{i+1}| \geq |s_i| + 1, \forall s_i \in S$.

Although $Ind(A, \varphi_i)$ is computed in linear time on the size of both strings, the computational complexity of the revision procedure depends on the number of strings generated by $Ind(A, \varphi_i), \forall \varphi_i \in \phi$. In some cases, $Ind(A, \varphi_i)$ may generate empty sets. However, in the worst case, the time complexity depends on the cardinality of the sets $L_i = \{x_1, \ldots, x_p\} = lit(\varphi_i) - lit(A), i = 1, \ldots, k$.

In order to improve the time complexity of our procedure, it is convenient to sort the clauses $\varphi_i \in \phi$ according to the cardinality of the sets L_i, $i = 1, \ldots, k$ from the smallest to the biggest and removing the clauses that are independent with A. Once the clauses are sorted in ϕ with respect to their cardinalities L_i, the operation $Ind(Ind(....Ind(Ind(A, \varphi_1), \varphi_2), \ldots, \varphi_k)$ is applied, determining so the succession: $S_0 = v(A)$, $S_1 = v(\varphi_1) - v(A)$, $S_2 = v(\varphi_2) - (v(\varphi_1) \cup v(A))$, $\ldots$, $S_k = v(\varphi_k) - (v(\varphi_{k-1}) \cup \ldots \cup v(\varphi_1) \cup v(A))$

Then, the number of new clauses in the worst case is given by:

$$| \, Ind(A, \phi) \, | \leq \prod_{i=1}^{k} | \, S_i \, | = | \, S_1 \, | * | \, S_2 \, | * \ldots * | \, S_{ik} \, | \, . \tag{2}$$

It is clear that the number of strings in $|Ind(A, \phi)|$ is not bigger than the number of assignments in $SAT(K) - FAL(\phi)$ since the falsifying assignments of $\varphi_i \in \phi$ are removed from the partial assignment denoted by the string A. That means $|S_1| * |S_2| * \ldots * |S_k| \in O(|SAT(K) - FAL(\phi)|)$. From this upper bound, we infer some tractable cases for ISAT.

Theorem 1. *Let K be a 2-CF formula and ϕ a CF formula. The following holds:*

1. *If $|SAT(K)| \leq poly(n)$ then ISAT is solved in polynomial time. In fact, we can have the set of models S explicitly and each model $s \in S$ can be checked: $\phi[s]$.*

2. *If $|SAT(K) - FAL(\phi)| \leq poly(n)$ then the number of strings in $|Ind(A, \phi)|$ is upper bounded by $|SAT(K) - FAL(\phi)| \leq poly(n)$.*
3. *When $\phi[T(x)]$ is false for all consistent $T(x)$, $Ind(K, \phi)$ finds the unsatisfiability of $(K \wedge \phi)$ in polynomial time on the set of literals of K and the number of clauses of ϕ. Consequently, this determines a tractable case for the 2-ISAT problem.*

Conclusions

We propose a novel method to review $K \vdash \phi$ when K and ϕ are both in Conjunctive Normal Forms. This initial method is extended to consider the incremental satisfiability (ISAT) problem. We have shown different cases where the ISAT problem can be solved in polynomial time.

Especially, we designed an algorithm for solving the 2-ISAT problem that allow us to determine an upper bound for the time-complexity of this problem. Furthermore, we have established some tractable cases for the 2-ISAT problem.

References

1. Buresh-Oppenheim J., Mitchell D., *Minimum 2CNF resolution refutations in polynomial time*, Proc. SAT'07 - 10th int. Conf. on Theory and applications of satisfiability testing, (2007), pp.300-313.
2. Cabodi G., Lavagno L., Murciano M., Kondratyev A., Watanabe Y., *Speeding-up heuristic allocation, scheduling and binding with SAT-based abstraction/refinement techniques*, ACM Trans. Design Autom. Electr. Syst., 15(2), (2010).
3. Creignou N., Papini O., Pichler R., Woltran S., *Belief Revision within fragments of propositional logic*, DBAI Tech. Report DBAI-TR-2012-75, (2012).
4. Eén N., S orensson K., *An Extensible SAT-solver*, In Enrico Giunchiglia and Armando Tacchella, editors, Selected Revised Papers of 6th International Conference on Theory and Applicationsof Satisfiability Testing (SAT), Santa Margherita Ligure, Italy, LNCS Vol. 2919, (2003), pp. 502-518.
5. Gallier J., *Logic for Computer Science: Foundations of Automatic Theorem Proving (chapter 9)*, (2003), online revision (free to download).
6. Gusfield, D., Pitt, L., A Bounded Approximation for the Minimum Cost 2-Sat Problem, *Algorithmica 8*, (1992), pp. 103-117.
7. Hooker J.N., Solving the incremental satisfiability problem. *Journal of Logic Programming 15*, (1993), pp.177-186.
8. Khardon R., Roth D.: Reasoning with Models, Artificial Intelligence 87, No.1, (1996), pp. 187-213.
9. Gutierrez J., Mali A., *Local search for incremental Satisfiability*, Proc. Int. Conf. on AI (IC-AI'02), Las Vegas, (2002), pp. 986-991.
10. Nadel A., Ryvchin V., Strichman O., *Ultimately Incremental SAT*, Proc. SAT 2014, LNCS Vol. 8561, (2014), pp. 206218.
11. Whittemore J., Joonyoung K., Sakallah K.A. *SATIRE: A New Incremental Satisfiability Engine*, In Proceedings of the 38th Design Automation Conference (DAC)-ACM, Las Vegas - USA, (2001), pp. 542-545.
12. Wieringa S., *Incremental Satisfbiability Solving and its Applications*, PhD thesis, Department of Computer Science and Engineering, Aalto University Pub., (2014).