Extending Model Checking to Efficient Propositional Inference

Guillermo de Ita Luna, Luis Polanco-Balcazar, and Omar Pérez-Barrios

***Abstract*—Propositional Inference is of special concern to Artificial Intelligence, and it has a direct relationship to automatic reasoning. Given a Knowledge Base Σ and a query Φ, propositional inference is concern to determine if Φ can be logically deduced from Σ, that is, if Σ ├ Φ.**

We show a deterministic and a complete polynomial time algorithm for given the knowledge base Σ in Disjunctive Form and Φ in Conjunctive Form, to decide if Σ ├ Φ.

***Index Terms*—Automatic reasoning, efficient propositional inference, knowledge base systems.**

I. Introduction

A widely accepted framework for reasoning in intelligent systems is the knowledge-based system approach [1]. The general idea is to keep the knowledge in some representation language with a well defined meaning assigned to those sentences. The sentences are stored in a Knowledge Base (KB) combined with a reasoning mechanism which is used to determine what can be inferred from the sentences in the KB.

Since logical, mathematical 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 [2]. Deductive propositional reasoning is usually abstracted as follows: Given a KB, assumed to capture our knowledge about the domain in question (the "world"), and a sentence Φ, a query that is assumed to capture the situation at hand, with both formulas expressed through propositional logic, decide whether KB implies Φ. This last problem is known as the *propositional entail problem*.

Let Σ be a KB and Φ be a query formula, we address here, a deterministic efficient procedure to decide if Σ ├ Φ. It is known that *logic entail* problem is a hard challenge in automatic reasoning and it is co-NP-Hard even in the propositional case [3]. Many other forms of reasoning which have been developed at least partly to avoid these computational difficulties, also have been shown to be hard to compute [4].

The propositional entail problem is one of the fundamental problems into automatic reasoning, and is a relevant task in many other issues, such as estimating the degree of belief, to review or update beliefs, abductive explanation, logical diagnosis, and many other procedures in Artificial Intelligence (AI) applications as planning, expert systems, approximate reasoning, etc. [3], [5]-[8].

As it has been pointed in [9]-[11], an important problem to explore is the computational complexity of the logical inference, and although the problem could be intractable for formulas in general, a precise determination of the complexity for procedures computing Σ ├ Φ has to be studied for classes of formulas Σ and Φ. And for propositional automatic reasoning, is essential to know under which restrictions for Σ and Φ, Σ ├ Φ could be checked in polynomial time.

We show here that the entail problem can be solved efficiently when Σ is in disjunctive form and Φ is in conjunctive form. This is an important case into automatic reasoning since many knowledge bases are considered to be in disjunctive forms, and then, to work with those classes of KB's allow efficient propositional entailment.

The research presented here continues the line pointed out by Eiter and many others [5]-[7], [9], [12], who have analyzed problems arising from deductive inference, such as searching for explanations, approximate reasoning, computing the degree of belief and incremental recompilation of knowledge. These works try to differentiate the classes of propositional formulas where such problems can be solved efficiently from those classes where such problems present an inherent exponential time complexity.

II. 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 $\neg x_i$. As usual, for each $x \in X$, $x^0 = \neg x$ and $x^1 = x$. We also denote $\neg x = \bar{x}$ as the negation of x.

A *clause* is a disjunction of different literals, we also consider a clause as a set of literals. For $k \in IN$, 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, and a *k-phrase* is a phrase with exactly k literals. A variable $x \in X$ *appears* in a clause (or phrase) c if either x or $\neg x$ is an element of c.

A *conjunctive form* (CF) is a conjunction of clauses, we also consider a CF as a set of clauses, while a *Disjunctive Form (DF)* is a disjunction of phrases. A k-CF is a CF containing only k-clauses. Similarly, a k-DF is a DF containing only k- phrases.

We say that a CF F is monotone if all of its variables appear with the same sign. A CF F with n variables represents a n-ary boolean function $F : \{0, 1\}^n \rightarrow \{0, 1\}$, although the same boolean function F has many equivalent representations and in particular, a CF as well as a DF is one of the way to represent any boolean function.

Manuscript received November 27, 2013; revised January 6, 2013. This work is partially supported by National Mexican System of Researchers (SNI) and Conacyt.

Guillermo de Ita Luna, Luis Polanco-Balcazar, and Omar Pérez-Barrios are with the Computer Science Faculty, Autonomus University of Puebla (FCC-BUAP), Mexico (e-mail: deita@cs.buap.mx, siulpolb@outlook.com, peb.omar@hotmail.com).

We use $\upsilon(Y)$ to express the variables involved in the object Y, where Y could be a literal, a clause, a phrase, a DF or a CF. For instance, for the clause $c = \{\neg x_1, x_2\}$, $\upsilon(c) = \{x_1, x_2\}$.

$Lit(F)$ is the set of literals, i.e. if $X = \upsilon(F)$, then $Lit(F) = X \cup \neg X = \{x_1, \neg x_1, ..., x_n, \neg x_n\}$. We use $Lit(Y)$ to express the literals involved in the object Y, where Y could be a clause, a phrase, a DF or a CF. We denote $\{1, 2, ..., n\}$ by $[n]$ and the natural number set by IN. We denote the cardinality of a set A by $|A|$.

An assignment s for F is a boolean function s: $\upsilon(F) \rightarrow \{0, 1\}$. An assignment s can also be considered as a set of non-complementary pairs of literals. If $l \in s$, being s an assignment, then s turns l *true* and $\neg l$ *false*.

Considering a clause c and an assignment s as a set of literals, c is *satisfied* by s if and only if $(c \cap s) \neq \emptyset$, and if for all $l \in c$, $l \in s$ then s falsifies c.

On the other hand, considering a phrase d also as a set of literals, and let s be an assignment over X, s satisfies d if $d \subseteq s$. And if for any literal $l \in d$, $l \in s$ then s falsifies d.

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 $\upsilon(F)$. Similarly, if $F_1 \subset F$, where F is a DF, then any assignment over $\upsilon(F_1)$ is a *partial* assignment over $\upsilon(F)$. Assuming $n = |\upsilon(F)|$ and $n_1 = |\upsilon(F_1)|$, any assignment over $\upsilon(F_1)$ has 2^{n-n1} extensions as assignments over $\upsilon(F)$.

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 contradicted by s. A model of F is an assignment defined on $\upsilon(F)$ that satisfies F.

If F is a DF, F is satisfied by an assignment s if any phrase in F is satisfied by s. F is contradicted by s if all phrase in F is falsified by s.

Given a formula F, let $S(F)$ be the set of all possible assignments defined over its set of variables $\upsilon(F)$. If $n = |\upsilon(F)|$ then $|S(F)| = 2^n$. We denote as $Sat(F)$ to the set of assignments from $S(F)$ which are models of F. $Fals(F)$ is the set of assignments from $S(F)$ which falsify F. For any propositional formula F, $S(F) = Sat(F) \cup Fals(F)$.

The SAT problem consists of determining if F has (or not) a model. The #SAT problem consists of counting the number of models of F.

If s is a model of F, it is denoted as $s \models F$. If an assignment s of $\upsilon(F)$ is not a model of F then s is a falsifying assignment of F.

A *Knowledge Base* (*KB*) Σ is a set of formulae. Given a *KB* Σ and a propositional formula Φ, we say that Σ *entails* Φ, denoted by $\Sigma \models \Phi$, if Φ is true for every model of Σ, i.e. $Sat(\Sigma) \subseteq Sat(\Phi)$.

III. Model-Checking for Propositional Inference

To prove $\Sigma \models \Phi$ is equivalent to show that $Sat(\Sigma) \subseteq Sat(\Phi)$. We extend the sets $\upsilon(\Phi)$ and $\upsilon(\Sigma)$ in order to build only one set containing all variables appearing in the formulas.

Let $X = \upsilon(\Sigma) \cup \upsilon(\Phi)$ be the set of variables, and let $Lit = X \cup \neg X$ be the set of literals appearing in Σ and Φ. We assume an order over the variables of X and that $n = |X|$, i.e. $X = \{x_1, x_2, \ldots, x_n\}$.

As $S(\Sigma) = S(\Phi) = Sat(\Phi) \cup Fals(\Phi)$, if $Sat(\Sigma) \subseteq Sat(\Phi)$ holds then $(Sat(\Sigma) \cap Fals(\Phi)) \subseteq (Sat(\Phi) \cap Fals(\Phi)) = \emptyset$. Indeed, to prove $\Sigma \models \Phi$ is equivalent to prove that

$$Sat(\Sigma) \cap Fals(\Phi) = \emptyset \quad (1)$$

If a KB Σ is in DF, i.e. $\Sigma = \vee_{i=1}^{m} \sigma_i$, where each σ_i, $i = 1, \ldots, m$ is a conjunction of literals, then it is easy to build $Sat(\Sigma)$, since each σ_i, $i = 1, \ldots, m$ determines a subset of satisfying assignments of Σ. In fact, $Sat(\Sigma) = \bigcup_{i=1}^{k} Sat(\sigma_i)$.

Also, if Φ is in CF, i.e. $\Phi = \wedge_{i=1}^{k} \varphi_i$, where each φ_i, $i = 1, \ldots, k$ is a disjunction of literals, then it is easy to build $Fals(\Phi)$ since each φ_i determines a subset of falsifying assignments of Φ, and actually, $Fals(\Phi) = \bigcup_{i=1}^{k} Fals(\varphi_i)$.

We exploit the previous relation to design a polynomial procedure to determine if Σ *entails* Φ. First, we show how to represent each $Sat(\sigma_i)$, $i = 1, \ldots, m$ *and each* $Fals(\varphi_i)$, $i = 1, \ldots, k$.

Let $\Phi = \wedge_{i=1}^{k} \varphi_i$ be a CF, then each φ_i, $i = 1, \ldots, k$ *is a* clause. For each $\varphi_i = \{l_{i1} \vee \ldots \vee l_{iki}\} \in \Phi$, let v_φ be a string such that its length is n.

The string v_φ is associated with $Fals(\varphi)$, and each one of its values: $v_\varphi[i]$, $i = 1, \ldots, n$ is determined, as:

$$v_\varphi[i] = \begin{cases} 0 & \text{if } x_i \in \varphi \\ 1 & \text{if } \neg x_i \in \varphi \\ * & \text{if neither } x_i \notin \varphi \text{ nor } \neg x_i \notin \varphi \end{cases} \quad (2)$$

We use the symbol $*$ to represent the elements that can take any truth value in the string v_φ, for example if $F = \{\varphi_1, \ldots, \varphi_m\}$ is a 2-CF, $n = |\upsilon(F)|$, $\varphi_1 = \{x_1, x_2\}$ and $\varphi_2 = \{x_2, x_3\}$ then we will write $v_{\varphi 1} = 00* * \ldots *$ and $v_{\varphi 2} = *00 * \ldots *$. This abuse of notation will allow us to give a concise and clear representation of the set $Fals(\varphi)$ in the rest of the paper, for considering the string v_φ as a string that represents the falsifying assignments of the clause φ.

v_φ represents in a succinct form all falsifying assignments of the clause φ, Since any assignment over X with values 0 or 1 in the same positions where v_φ has those, and with any value $\{0, 1\}$ in the positions where v_φ has $*$, is a falsifying assignment for φ. we call *falsifying string* to such string v_φ that represents the falsifying assignments of φ. In fact, v_φ represents the subset of $2^{n-|\varphi|}$ falsifying assignments of φ.

On the other hand, let $\sigma = (l_1 \wedge \ldots \wedge l_j)$ be a phrase defined over $Lit(X)$. 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:

$$v_\varphi[i] = \begin{cases} 0 & \text{if } \neg x_i \in \varphi \\ 1 & \text{if } x_i \in \varphi \\ * & \text{if neither } x_i \notin \varphi \text{ nor } \neg x_i \notin \varphi \end{cases} \quad (3)$$

Similarly, the string v_σ is a succinct form to represent $Sat(\sigma)$. Because any assignment over X with values 0 or 1 in the same

positions where v_σ has those, and with any value {0, 1} in the positions where v_σ has $*$, is a satisfying assignment for σ. Thus, v_σ represents the set of $2^{n-|\sigma|}$ satisfying assignments of the phrase σ. We call to v_σ the *satisfying string* for the phrase σ.

As our procedure exploit the relations: $Sat(\Sigma) = \bigcup_{i=1}^{m} Sat(\sigma_i)$, and $Fals(\Phi) = \bigcup_{i=1}^{k} Fals(\varphi_i)$, to reduce sizes of CF's and DF's is a relevant task in order to build efficient algorithms for the following goals in automatic deduction as to check if $\Sigma \vdash \Phi$.

It is common to review a formula in order to reduce its size keeping just the necessary subformulas in F. For example, for a CF it is common to delete all redundant clauses as: tautological clauses and clauses with pure literals. The application of the following rules allow to reduce the size of CF's an DF's.

A. Rule of Pure Literal

Let F be a CF, $l \in Lit(F)$ is a pure literal if l appears in F but $\neg l$ does not appear in F.

If a clause contains a pure literal, that clause can be eliminated from F, keeping the logical value of F. Because if the literal l is set to *True*, the clause containing l is also *True*, and then it can be deleted from F. Similarly, if a formula G is in DF, any phrase containing a pure literal can be falsified by set *False* to that pure literal and the phrase is also *False*, then the phrase can be eliminated from G.

Other relevant rules to reduce sizes of CF's and DF's are subsumed clauses and subsumed phrases rules.

B. Subsumed Clause Rule

Given two clauses c_i and c_j of a CF F, if $Lit(c_i) \subseteq Lit(c_j)$ then c_j is subsumed by c_i, and c_j can be deleted from F. Because all satisfying assignment of c_j is a satisfying assignment of c_i, that is $Sat(c_j) \subseteq Sat(c_i)$. Thus, it is enough to keep just c_i (the clause which subsumes) in the CF.

C. Subsumed Phrase Rule

Given two phrases d_i and d_j of a DF F, if $Lit(d_i) \subseteq Lit(d_j)$ then d_j is subsumed by d_i, and d_j can be deleted from F. Because all falsifying assignment of d_j is a falsifying assignment of d_i, that is $Fals(d_j) \subseteq Fals(d_i)$. Thus, it is enough to keep just d_i (the phrase which subsumes) in the DF.

IV. Polynomial Algorithm

Assuming that Σ is in DF and Φ in CF, i.e. $\Sigma = \vee_{i=1}^{m} \sigma_i$, and $\Phi = \wedge_{i=1}^{k} \varphi_i$, then $Sat(\Sigma) = \bigcup_{i=1}^{m} Sat(\sigma_i)$, and similarly, $Fals(\Phi) = \bigcup_{i=1}^{k} Fals(\varphi_i)$. Furthermore, we have an easy way to represent each $Sat(\sigma_i)$ and each $Fals(\varphi_j)$ based on the strings $v_{\sigma i}$ and $v_{\varphi j}$. The key point in our procedure is to check if it is possible to combine the assignments of both strings to form a valid set of assignments on X, and such new combined string will satisfy a $\sigma_i \in \Sigma$ and falsify a $\varphi, \in \Phi$, proving so that Φ is not inferred from Σ.

Thus to prove that $\Sigma \vdash \Phi$, it is equivalent to prove that $Sat(\Sigma) \cap Fals(\Phi) = \emptyset$, and it is equivalent to show that

$$Sat(\sigma_i) \cap Fals(\varphi_j) = \emptyset, \ \forall i = 1,\ldots,m, \ j = 1,\ldots k \qquad (4)$$

Then, we have to build the sets of assignments $Sat(\sigma_i)$ and $Fals(\varphi_j)$, for each $\sigma_i \in \Sigma$ and for all $\varphi_j \in \Phi$. And for this, we take advantages of the succinct form to represent those sets via the strings shown in previous section. In fact, we need just the literals associated with the fixed values appearing in both strings v_σ and v_φ.

The procedure *Inference* checks if there exist any $\sigma_i \in \Sigma$ and any $\varphi_j \in \Phi$ such that $Sat(\sigma_i) \cap Fals(\varphi_j) \neq \emptyset$ and in this case, it outputs *False* indicating that $\Sigma \vdash \Phi$ does not hold. Otherwise, it has proved that $Sat(\Sigma) \cap Fals(\Phi) = \emptyset$, and therefore $\Sigma \vdash \Phi$.

Algorithm 1: Procedure *Inference* (Σ, Φ)

Input: Σ = {A Knowledge Base}, Φ = {New Knowledge}
Output: *True/False* = $\Sigma \vdash \Phi$ / $\Sigma \vdash \Phi$
for all $\sigma_i \in \Sigma$ **do**
  $A = Lit(\sigma_i)$; {A satisfies σ_i}
  for all $\phi_j \in \Phi$ **do**
    $B = Lit(\neg\varphi_j)$; {B satisfies φ_i }
    $s = A \cup B$; {s could be a valid assignment or not}
    if (*no_comp_literals in*(s)) **then**
      Returns(*False*) {*s* an assignment, $s(\Sigma) = 1$, and $s(\Phi) = 0$}
    end if
  end for
end for
Returns(*True*)

Notice that A and B in the procedure Inference represents in fact, a subset of assignments. And the union $s = A \cup B$ could be (or not) a valid set of assignments. For example, if there is a literal l such that $l \in s$ and $\neg l \in s$ then s does not represent a valid subset of assignments. And that last property is checked via the function *no_comp_ literals_in*(*s*).

Notice also that the implementation of the union $A \cup B$ and the function *no_comp_literals_in*(*s*) can be done in efficient way according to the representation of a set. But in general, both operations can be done in linear time complexity on the number of maximum elements on the set, that in this case is of order O(n), $n = |X|$.

V. Soundness, Completeness and Time Complexity of the Procedure

Let $\Sigma = \vee_{i=1}^{m} \sigma_i$ be a CF and let $\phi = \wedge_{i=1}^{k} \varphi_i$ be a DF, where $\sigma = (l_{i1} \wedge \ldots \wedge l_{im_i})$ and $\varphi = (l_{j1} \wedge \ldots \wedge l_{jk_j})$. We analyze here the algorithm *Inference* which decides if $\Sigma \vdash \Phi$.

A. Proof (Soundness)

If the procedure *Inference* outputs *True*, then effectively $\Sigma \vdash \Phi$.

Proof :

Inference outputs *True* if and only if there is a pair of

complementary literals in $s = Lit(\sigma_i) \cup Lit(\overline{\varphi_j})$, for all $\sigma_i \in \Sigma$ and $\varphi_j \in \Phi$.

It means that there exists a literal l in any possible assignment satisfying Σ and falsifying Φ, such that $l \in s$ and $\neg l \in s$, but none valid assignment could contain at the same time, a pair of complementary literals.

Thus, there does not exist an assignment s with $s(\Sigma) = True$ and $s(\Phi) = False$, therefore $Sat(\Sigma) \cap Fals(\Phi) = \emptyset$ and then $\Sigma \vdash \Phi$.

B. Proof (Completeness)

Assume that $\Sigma \vdash \Phi$, then *Inference*(Σ,Φ) outputs *True*.

Proof :

As $\Sigma = \vee_{i=1}^{m} \sigma_i$, then $Sat(\Sigma) = \bigcup_{j=1}^{m} Sat(\sigma_i)$, similarly $Fals(\Phi) = \bigcup_{j=1}^{k} Fals(\varphi_i)$ because $\Phi = \wedge_{i=1}^{k} \varphi_i$.

Let A be a valid assignment from $Sat(\Sigma)$, then there exist σ_i in Σ such that $A(\sigma_i) = 1$, given that $Sat(\Sigma) = \bigcup_{j=1}^{m} Sat(\sigma_i)$.

Let B a valid assignment for $Fals(\Phi)$, then there exist $\varphi_j \in \Phi$ such that $B \in Fals(\varphi_j)$ because $Fals(\phi) = \bigcup_{j=1}^{k} Fals(\varphi_i)$.

Let $s = A \cup B$. If s has not complementary literals then s is a valid assignment over X, and $s \in Sat(\Sigma)$, as well as $s \in Fals(\phi)$, but in this case *Inference* outputs *False* because the procedure *no_comp_literals_in*(*s*) holds. Then, if $\Sigma \vdash \Phi$ such assignment s does not exist and *Inference* must output *True*.

C. Time Complexity

Inference involves two for's, one of size $|\Sigma|$ and the other of size $|\Phi|$, then it performs of order $O(|\Sigma| \cdot |\Phi|)$ operations; union between two sets with n elements at most, and a revision for complementary members on a set.

Both set operations (union and revision of members) are performed in linear time complexity according with the maximum number of elements in the sets, that is $n = |X|$.

Then, the total time complexity in the worst case is the order $O(|\Sigma| \cdot |\Phi| \cdot |X|)$. Indeed Inference is a linear time deterministic algorithm on the size of its inputs: Σ and Φ, and on the number of variables appearing in both formulas $X = \upsilon(\Sigma) \cup \upsilon(\Phi)$.

In order to impact the time complexity of Inference procedure, the application of pure literals, subsumed clauses and subsumed phrases rules would be beneficial, because those rules can reduce the real sizes $|\Sigma|$ and $|\Phi|$. Although, the application of those rules also implies a cost on the time of pre-processing the input formulas: Σ and Φ.

According to the computational representation of the formulas Σ and Φ, both rules can be applied very efficiently. For example, the representation of Σ and Φ using indexes over a fix set of variables X, produces algorithms with a reduced time complexity.

On the other hand, the literal pure rule implies to look for a literal (and its complementary value) on the size of the formulas. Then, literal pure rule can be implemented with a time complexity, in the worst case, of $O(|X| \cdot |\Sigma|)$ and $O(|X| \cdot |\Phi|)$, when it is applied on Σ and Φ, respectively.

And the subsumed clause rule (subsumed phrase rule) requests that each clause (or phrase) will be compared with the rest of the clauses (phrases) in the formula in order to look for subset of literals. That implies the order of $O(|X| \cdot |\Sigma|^2)$ and $O(|X| \cdot |\Phi|^2)$ basic operations in the worst case, when they are applied on Σ and Φ, respectively.

In whatever case, those pre-processing procedures have polynomial time cost, and usually they are done off-line, optimizing the time cost of working on-line for deciding if $\Sigma \vdash \Phi$.

VI. Conclusions

A fundamental problem in deductive propositional reasoning is the entail Problem, i.e. given a KB Σ and a query formula Φ to decide if $\Sigma \vdash \Phi$.

We have shown that the entail problem is solved efficiently when Σ is in disjunctive form and Φ is in conjunctive form. In fact, we show a linear time procedure on the size of its inputs: Σ, Φ and on the number of variables involved in both formulas.

To design an efficient procedure for the entail problem has repercussions in the area of automatic reasoning. Thus, we have presented an important case of efficient propositional entailment. Since many knowledge bases are considered to be in disjunctive forms, our algorithm could provide efficient automatic reasoning schemes.

References

[1] J. McCarhy, "Programs with common sense," in *Proc. the Symposium on the Mechanization of Thought Processes*, vol. 1, 1958, pp. 77-84.

[2] N. Shankar, *Metamathematics, Machines, and Gödel's Proof*, Cambridge Tracks in Theoretical Computer Science No. 38, Cambridge University Press, 1997.

[3] R. Khardon and D. Roth, "Reasoning with Models," *Artificial Intelligence*, vol. 87, no.1, pp. 187-213, 1996.

[4] B. B. Selman, "Tractable default reasoning," PhD thesis, Department of Computer Science, University of Toronto, 1990.

[5] A. Darwiche, "On the tractable counting of theory models and its application to truth maintenance and belief revision," *Jour. of Applied Non-classical Logics*, vol. 11, pp. 11-34, 2001.

[6] T. Eiter, M. Fink, G. Sabattini, and H. Thompits, "Considerations on up- dates of logic programs," *JELIA00, Lec. Notes in Artificial Intelligence*, vol. 1, 2000.

[7] G. Gogic, C. Papadimitriou, and M. Sideri, "Incremental recompilation of knowledge," *Jour. of Artificial Intelligence Research*, vol. 8, pp. 23- 37, 1998.

[8] D. Roth, "On the hardness of approximate reasoning," *Artificial Intelligence*, vol. 82, pp. 273-302, 1996.

[9] T. Eiter and G. Gottlob, "On the complexity of propositional knowledge base revision, updates, and counterfactuals," *Artificial Intelligence*, vol. 57, 1992.

[10] H. Katsuno and A. Mendelzon, "On the difference between updating a knowledge base and revising it," in *Proc. KR-91*, 1991, pp. 387-395.

[11] P. Liberatore and M. Schaerf, "The complexity of model checking for belief revision and update," in *Proc. Thirteenth Nat. Conf. on Art. Intelligence (AAAI96)*, 1996.

[12] B. B. Zanuttini, "New polynomial classes for logic-based abduction," *Journal of Artificial Intelligence Research*, vol. 19, pp. 1-10, 2003.

Guillermo de Ita did his BS in computer science in the Faculty of Computer Sciences in the Autonomus University of Puebla (BUAP), Mexico. The master and Ph. D. program in electrical engineering in the Cinvestav - I.P.N, Mexico. He has worked by 10 years as a developer and consulter for Database Systems and Geographic Information Systems for different enterprises in Mexico. He has done researching stances in Chicago University, Texas A&M, INAOEP Puebla, and in the INRIA Institute in

Lille1 - France. Currently, he is a senior researcher - professor of the Faculty of Computer Sciences in the BUAP.

Omar Pérez did his BS in computer science in the Faculty of Computer Sciences of the Autonomous University of Puebla (BUAP), Mexico. He is interested in researching of propositional inference, automatic reasoning and artificial intelligence.

Luis Polanco did his BS in computer science in the Faculty of Computer Sciences in the Autonomous University of Puebla (BUAP), Mexico. He is interested in researching of propositional inference, automatic reasoning and artificial intelligence.