ON-LINE ALGORITHMS FOR POLYNOMIALLY SOLVABLE SATISFIABILITY PROBLEMS

GIORGIO AUSIELLO* AND GIUSEPPE F. ITALIANO[†]

▷ Given a propositional Horn formula, we show how to maintain on-line information about its satisfiability during the insertion of new clauses. A data structure is presented which answers each satisfiability question in $O(1)$ time and inserts a new clause of length q in $O(q)$ amortized time. This significantly outperforms previously known solutions of the same problem. This result is extended also to a particular class of non-Horn formulae already considered in the literature, for which the space bound is improved. Other operations are considered, such as testing whether a given hypothesis is consistent with a satisfying interpretation of the given formula and determining a truth assignment which satisfies a given formula. The on-line time and space complexity of these operations is also analyzed. ◁

1. INTRODUCTION

In several so-called knowledge based applications, knowledge is represented by means of logical formulae. In order to allow efficient algorithms for deduction and consistency checking in such applications, particular classes of formulae of predicate calculus have been often considered. Among them, Horn formulae are particularly interesting in view of the fact that other knowledge-representation formalisms such as and-or graphs and production rules have essentially the same syntactic and semantic properties [12].

While designing knowledge based systems (in the sequel referred to as KBSs), one perceives the need of performing on line several operations [2, 13], such as to

*Partially supported by MPI Italian National Project on Theory and Design of Algorithms.
[†]Partially supported by an IBM Graduate Fellowship.

Address correspondence to Giuseppe F. Italiano, Department of Computer Science, Columbia University, New York, NY 10027.

Received January 1988; accepted March 1989.

insert a new clause into a formula, to verify whether a formula is satisfiable or whether a given interpretation satisfies a formula, etc. In [2] the general problem of updating a KBS is considered in a rather general framework. Insertion and deletion operations on the knowledge base are defined, but no effort is made to provide an evaluation of the time and space required by the proposed management procedures. This approach was extended in [13], where the concept of stratification was applied to disjunctive knowledge bases. This made it possible to define classes of algorithms for interactively constructing models of a KBS expressed in logic. Due to the general point of view, the computational complexity of such algorithms is mostly exponential in the size of the input.

In this paper, in order to approach efficiency issues in the interactive management of a KBS, we tackle the more precise problem of maintaining simple propositional formulae such as Horn formulae and some disjunctive non-Horn formulae already studied in the literature, for which the satisfiability problem has been shown to be polynomially solvable [6, 9, 11, 19]. The approach followed in this paper, moving from the previous work in [6, 9], is aimed at considering sequences of operations such as those described above, and at designing efficient algorithms and data structures for supporting them on line. In this framework, our attention will be devoted to analyzing the amortized cost [17] over a whole sequence of operations.

A similar viewpoint has been taken in the study of dynamic graphs [7, 8, 10, 16]. By representing Horn formulae by means of directed hypergraphs [4], we are able to apply the same algorithmic ideas and to provide results which may be useful both for application to logic formulae and per se, as an extension to directed hypergraphs of the results known for dynamic graphs.

The main result of the paper is to show how to insert a clause of size q into a Horn formula in $O(q)$ amortized time in such a way that the satisfiability of the whole formula can be checked at any time in $O(1)$. This outperforms by an order of magnitude the best known algorithms for the same problem [6, 9], which can require even $O(n)$ time for testing the satisfiability of the formula each time a new clause is inserted, where n is the total length of the formula. Put in other words, we exhibit one algorithm which takes a total of $O(n)$ time in the worst case in maintaining information about the satisfiability of a Horn formula of length n during the dynamic insertion of m clauses, while the previous known algorithms [6, 9] require $O(mn)$ worst-case time for the same problem.

Besides, it is shown that the same operations may be performed on a particular class of non-Horn formulae in such a way that the overall cost of a sequence of operations is at most quadratic in the total size of the input. The required amount of storage is in both cases linear in the size of the input, which improves the previously known results for the class of non-Horn formulae [1].

Finally, it may be observed that the proposed representation may efficiently support other operations (such as for example implications among propositional variables), provided that a suitable amount of extra storage is allowed.

The remainder of the paper is organized as follows. In Section 2 some basic terminology about propositional Horn formulae and directed hypergraphs is introduced. In Section 3 the hypergraph formalism is used as a powerful tool for designing on-line algorithms for the satisfiability of Horn formulae. These results are extended in Section 4 to other on-line operations on propositional Horn formulae, and in Section 5 to a class of propositional formulae which properly includes Horn formulae. In Section 6 we make some concluding remarks.

2. HORN FORMULAE AND DIRECTED HYPERGRAPHS

Directed hypergraphs [4] are a generalization of directed graphs which may be useful for representing structures and concepts in several areas of computer science such as rewriting systems, problem solving [14], Petri nets [15], and functional dependency in relational databases [3, 4, 18]. In this section we shall see how the hypergraph formalism may be a powerful tool also in dealing with propositional Horn formulae. We begin by giving some preliminary definitions about both directed hypergraphs and propositional Horn formulae. Next, we show that the satisfiability of a Horn proposition is expressible as a hyperpath problem on a properly defined directed hypergraph.

A *directed hypergraph* $\boldsymbol{H}$ is a pair $\langle N, H\rangle$, where N is the set of nodes and H is the set of *hyperarcs*. Each hyperarc is an ordered pair (X, i) such that X is a nonempty subset of N and $i \in N$. X is said to be the source set of the hyperarc (X, i). Directed hypergraphs are henceforth called hypergraphs.

Given a hypergraph $\boldsymbol{H} = \langle N, H\rangle$, a nonempty subset of nodes $X \subseteq N$, and a node $i \in N$, there is a *(directed) hyperpath* from X to i if one of the following conditions holds:

(1) $i \in X$ *(extended reflexivity)*, or

(2) there exists a set of nodes $Y = \{y_1, y_2, \ldots, y_q\}$ such that (Y, i) is a hyperarc in H and for $j = 1, 2, \ldots, q$ there is a hyperpath from X to y_j *(extended transitivity)*.

The *rank* r of a hyperpath from X to i is defined inductively as follows. If i belongs to X, then $r = 0$. Otherwise, $r = 1 + \max\{r_1, r_2, \ldots, r_q\}$, where r_j, $1 \leq j \leq q$, is the rank of the hyperpath from X to y_j.

Example 2.1. The hypergraph $\boldsymbol{H} = \langle\{A, B, C, D, E, F, G\}, \{(ABC \rightarrow D), (CD \rightarrow E), (A \rightarrow F), (D \rightarrow F), (E \rightarrow G)\}\rangle$ is shown in Figure 1.

In order to approach hypergraph problems, a graphical representation has been introduced in [3]. Given a hypergraph $\boldsymbol{H} = \langle N, H\rangle$, the *FD graph* of $\boldsymbol{H}$ is the labeled graph $G(\boldsymbol{H}) = \langle N_H, A_f, A_d\rangle$, where:

$N_H = N \cup N_c$ is the set of nodes, where N is called the set of *simple nodes* and $N_c = \{X \subseteq N \mid X \text{ is a source set in } \boldsymbol{H}\}$ is called the set of *compound nodes*;

$A_f \subseteq N_H \times N$ is the set of arcs (referred to as *full arcs*)

$$\{(X, i) \mid \text{there exists a hyperarc } (X, i) \text{ in } \boldsymbol{H}\};$$

$A_d \subseteq N_c \times N$ is the set of arcs (referred to as *dotted arcs*)

$$\{(X, j) \mid X \in N_c \text{ and } j \in X\}.$$

The name FD graph stands for *functional-dependency* graph, since FD graphs were first introduced in order to represent functional dependency in database schemes [3].

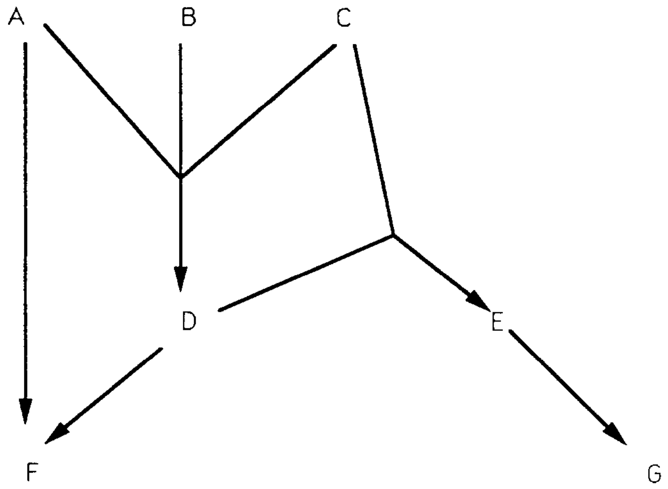

FIGURE 1. The hypergraph given in Example 2.1.

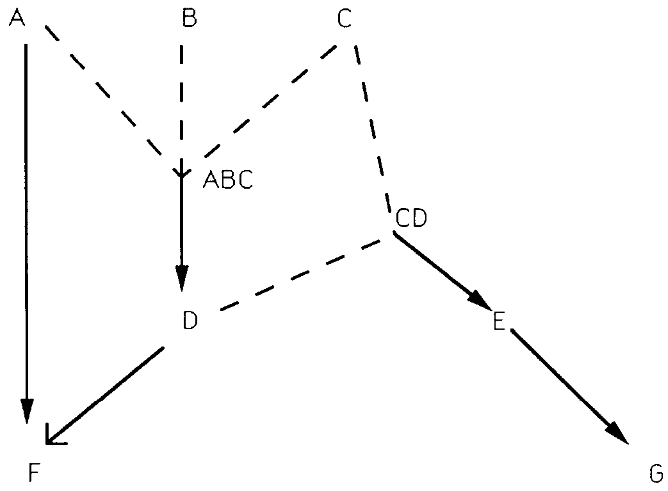

FIGURE 2. The FD-graph representation of the hypergraph given in Figure 1.

Example 2.2. The FD-graph representation of the hypergraph of Figure 1 is given in Figure 2.

The notion of hyperpath can be reformulated on the corresponding FD graph, giving rise to the concept of *FD path* (see for example [4]).

Several structures used for modeling problems in computer science may be represented by means of directed hypergraphs or (equivalently) by means of FD graphs. In particular, this happens with a class of logical formulae known as Horn formulae, which may be defined as follows.

A *literal* is either a propositional variable P (a *positive literal*) or the negation $\neg P$ of a propositional variable P (a *negative literal*). A *Horn clause* C_i is a disjunction of literals with at most one positive literal. A *Horn formula* A is a conjunction of Horn clauses, i.e. $A = C_1 \wedge C_2 \wedge C \cdots \wedge C_m$.

It is easy to see that Horn clauses can be only of three types:

(1) Q (a propositional variable), or

(2) $\neg P_1 \vee \neg P_2 \vee \cdots \vee \neg P_q$, $q \geq 1$ ($P_1, P_2, \ldots, P_q$ are distinct propositional variables), or

(3) $\neg P_1 \vee \neg P_2 \vee \cdots \vee \neg P_q \vee Q$, $q \geq 1$.

In the remainder of the paper we shall denote by k the number of distinct propositional variables in a given Horn formula, by m the number of Horn clauses, and by n the total length of the Horn formula (i.e., the total number of occurrences of literals).

We associate to a propositional Horn formula A a directed hypergraph H_A, referred to as the *hypergraph corresponding to* A, defined as follows.

Definition 2.1. Given a Horn formula $A = C_1 \wedge C_2 \wedge \ldots \wedge C_m$, H_A is a directed hypergraph whose nodes correspond to each propositional variable occurring in A, plus two extra nodes for *true* and *false*. The hyperarcs correspond to the Horn clauses, according to the following rules:

(i) for each Horn clause which consists of only a positive literal Q, there is a hyperarc from $\{true\}$ to Q;

(ii) for each Horn clause of the type

$$\neg P_1 \vee \neg P_2 \vee \cdots \vee \neg P_q, \qquad q \geq 1,$$

there is a hyperarc from the source set $X = (P_1, P_2, \ldots, P_q)$ to *false*;

(iii) for each Horn clause of the type

$$\neg P_1 \vee \neg P_2 \vee \cdots \vee \neg P_q \vee Q, \qquad q \geq 1,$$

there is a hyperarc from the source set $X = (P_1, P_2, \ldots, P_q)$ to Q.

Example 2.3.

$$H = (B) \wedge (\neg A \vee \neg B \vee \neg C \vee D) \wedge (\neg A \vee F)$$
$$\wedge (\neg D \vee F) \wedge (\neg C \vee \neg D \vee E) \wedge (\neg E \vee G) \wedge (\neg G).$$

The hypergraph corresponding to the Horn formula H is exhibited in Figure 3.

Notice that a correspondence may be shown between such hypergraphs and the Horn labeled digraphs G_A defined in [6]. In particular, they are defined on the same set of nodes N, and for each hyperarc (X, Q) of the source set $X = (P_1, P_2, \ldots, P_q)$ in H_A corresponding to the ith clause in A, there are q arcs labeled with i from P_j, $1 \leq j \leq q$, to Q in G_A. This gives rise to the following theorem.

Theorem 2.1. Given a Horn formula A, let $H_A = \langle N, H \rangle$ be its corresponding hypergraph. Then:

(i) *A is satisfiable if and only if there is no hyperpath from true to false in H_A.*

(ii) *If for some propositional variable Q there is a hyperpath from true to Q in H_A, then Q must have truth value true in any satisfying interpretation of A.*

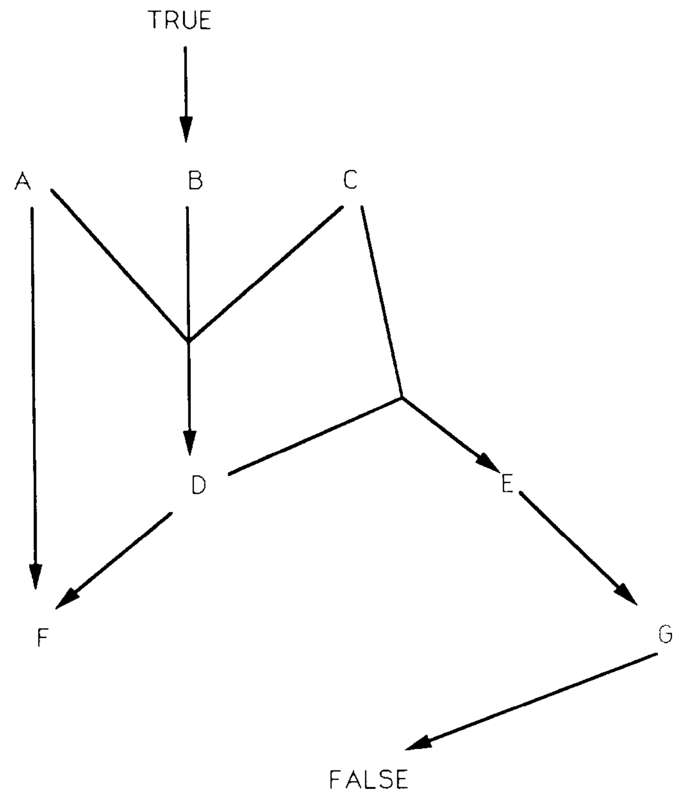

FIGURE 3. The hypergraph corresponding to the Horn formula given in Example 2.3.

Proof. It is a straightforward consequence of Theorems 2 and 3 in [6]. □

If we think of {*false*,*true*} as a boolean algebra in which *false* < *true*, then also the cartesian product $\{false,true\}^k$ is a boolean algebra. Then we have the following corollary, whose simple proof has been omitted.

Corollary 2.1. Given a satisfiable Horn formula A, let H_A be its corresponding hypergraph. The truth assignment v such that $v(P_i) = true$ if and only if there is a hyperpath from the node true to P_i, and false otherwise, is the least truth assignment in the boolean algebra $\{false,true\}^k$ satisfying A.

We are now able to characterize in graph-theoretical terms the satisfying truth assignments of a given Horn formula. First, we need some preliminary results.

Lemma 2.1. Let A be a satisfiable Horn formula, H_A its corresponding hypergraph, and X any set of variables. If there is a satisfying interpretation v of A in which the variables in X get truth value true, then there will be no hyperpath from any subset of $X \cup \{true\}$ to a node Q for which $v(Q) = false$.

Proof. We proceed by induction on the rank d of hyperpaths ($d > 0$, since the case $d = 0$ cannot happen).

No hyperpaths of rank $d = 1$ can exist, since the presence of a hyperarc from $S \subseteq X \cup \{true\}$ to *false* would imply a clause

$$\neg P_1 \vee \neg P_2 \vee \cdots \vee \neg P_q$$

with $P_i \in X$ $(1 \le i \le q)$, which would not be satisfied in the interpretation v.

Suppose now that no hyperpaths of rank less than l exist from $X \cup \{true\}$ to nodes which get truth value *false* in the interpretation v, and assume by contradiction that there is a hyperpath of rank l from $X \cup \{true\}$ to a node Q such that $v(Q) = false$. Then, by definition of hyperpath, there will be a hyperarc from $P_1, P_2, \ldots, P_q$ to Q and q hyperpaths of rank less than l from $X \cup \{true\}$ to P_i $(1 \le i \le q)$. The hyperarc from $P_1, P_2, \ldots, P_q$ to Q implies a clause

$$\neg P_1 \vee \neg P_2 \vee \cdots \vee \neg P_q \vee Q,$$

which has to be satisfied by the interpretation v. Since $v(Q) = false$, there will be an integer j $(1 \le j \le q)$ such that $v(P_j) = false$, thus contradicting the inductive hypothesis.

This completes the induction step and gives the lemma. $\square$

Theorem 2.2. Let A be a Horn formula and $\boldsymbol{H}_A$ be its corresponding hypergraph. Given a set of variables X, there exists a satisfying interpretation v of A in which the variables in X get truth value true if and only if there is no hyperpath in $\boldsymbol{H}_A$ from $X \cup \{true\}$ to false.

PROOF. *"If" part*: A hyperpath from $X \cup \{true\}$ to *false* would imply a clause

$$C = \neg P_1 \vee \neg P_2 \vee \cdots \vee \neg P_q$$

and q hyperpaths from $X \cup \{true\}$ to P_l $(1 \le i \le q)$. According to Lemma 2.2, $v(P_i) = true$ $(1 \le i \le q)$, which would contradict the satisfiability of the interpretation v.

"Only if" part: If there is no hyperpath from $X \cup \{true\}$ to *false*, then consider the following set:

$$Y = \{Q \in V \mid \text{there is a hyperpath from } X \cup \{true\} \text{ to } Q\}.$$

Let us define the following interpretation:

$$v(P) = \begin{cases} true & \text{if and only if} \quad P \in Y, \\ false, & \text{otherwise.} \end{cases}$$

Since $X \subseteq Y$, the variables in X get truth value in v. Furthermore, we can have three types of clauses:

(i) Q,

(ii) $\neg P_1 \vee \neg P_2 \vee \cdots \vee \neg P_q$,

(iii) $\neg P_1 \vee \neg P_2 \vee \cdots \vee \neg P_q \vee Q$.

Type (i) clauses are satisfied by v, since for such a clause there is a hyperarc from the node *true* to Q and thus $v(Q) = true$.

Also type (ii) clauses are satisfied by v, since for such a clause there is a hyperarc from $\{P_1, P_2, \ldots, P_q\}$ to *false* and we cannot have $P_i \in Y$ for each i (otherwise we would have a hyperpath from $X \cup \{true\}$ to *false*).

Type (iii) clauses imply a hyperarc from $\{P_1, P_2, \ldots, P_q\}$ to Q. If $Q \in Y$, then it makes the clause satisfied. Otherwise, at least one variable in $\{P_1, P_2, \ldots, P_q\}$ must not belong to Y (otherwise also $Q \in Y$), thus again satisfying the clause. □

As a straightforward consequence, we state the following corollary, which will be useful in the sequel.

Corollary 2.2. Given a satisfiable Horn formula A, let H_A be its corresponding hypergraph. For any variable P, there exists a satisfying interpretation of A in which P gets truth value true if and only if there is no hyperpath in H_A from $\{P$,true$\}$ to false.

3. ON-LINE ALGORITHMS FOR THE SATISFIABILITY OF PROPOSITIONAL HORN FORMULAE

Algorithms for testing the satisfiability of Horn formulae have been developed by several researchers in recent years [6, 9, 11]. A common feature of the above algorithms is that they are all *off-line*, i.e., they require that complete information about the Horn formula be available in advance.

In some applications, however, it is desirable to develop algorithms which receive one Horn clause at a time and allow fast queries about the satisfiability of the whole formula so far received. Such algorithms, which are required to complete each operation before the next one is known, are called *on-line*. This situation arises for example when a rule based system is being built and the corresponding set of rules dynamically changes by means of either the insertion of new rules or the elimination of rules which caused inconsistency [2, 13]. In many cases, on-line algorithms are less efficient on the whole sequence of operations than the corresponding off-line algorithms, since some price must generally be paid to acquire the on-line property.

In what follows, we shall see how the on-line satisfiability problem for Horn formulae can be efficiently solved. In more detail, we define the following two operations on Horn formulae:

satisfy(A): check the satisfiability of the Horn formula A;

insert(c, A): add the Horn clause c to the Horn formula A, giving rise to the new Horn formula $A \wedge c$.

We describe now a data structure in which each *satisfy* is performed in $O(1)$ worst-case time and which is able to insert a clause of size q in $O(q)$ amortized time. As a result, the total time involved in maintaining the data structure on line during the insertions of m clauses is $O(n)$. This bound improves by an order of magnitude the $O(mn)$ time required by the best known algorithms [6, 9] for the same problem.

The idea underlying the data structure is to deal with the hypergraph H_A corresponding to the Horn formula A. To be more precise, $G(H_A)$, the FD graph of H_A, is maintained. Simple nodes correspond to the variables of the formula A, while for each clause in A a compound node is introduced as shown in Figure 4, according to Definition 2.1.

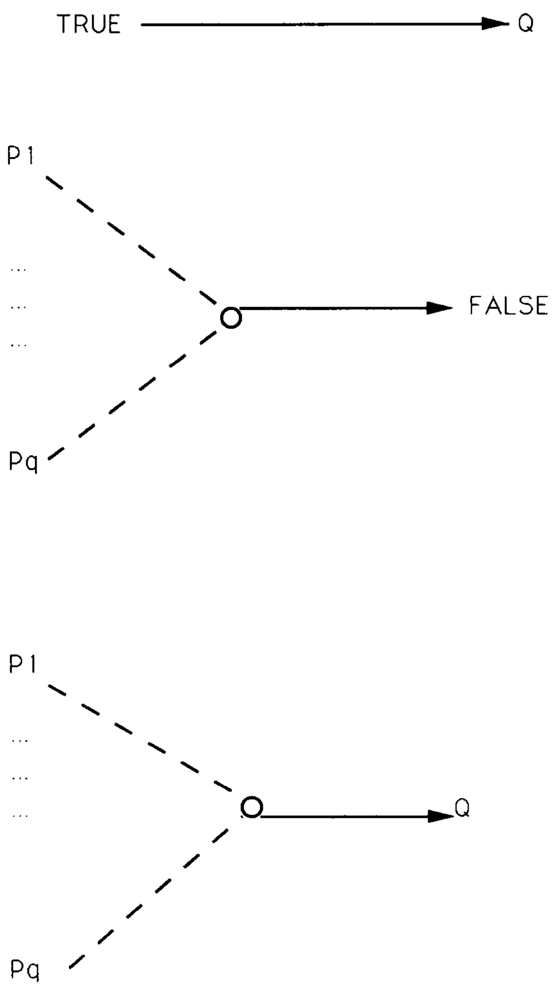

FIGURE 4. The correspondence between Horn clauses and hyperarcs.

Note that in such a representation only full arcs can enter a simple node, while compound nodes are accessed by means of dotted arcs. Moreover, there is only one (full) arc leaving each compound node.

Owing to Theorem 2.1, queries about the satisfiability of a Horn formula A can be answered by simply checking the presence of an FD path (from the node *true* to the node *false*) in the FD graph $G(H_A)$. In order to speed up such satisfiability queries, we maintain the closure of the node *true* by associating to each simple node $i \in N$ an integer, referred to as *status*(i), defined as follows:

$$\mathit{status}(i) = \begin{cases} 0 & \text{if there is an FD path in } G(H_A) \text{ from } \mathit{true} \text{ to } i, \\ 1 & \text{otherwise.} \end{cases}$$

With this additional information, the satisfiability of a Horn formula A can be checked by examining the status of the node *false* in the corresponding FD graph. More precisely, A is satisfiable if and only if *status*(*false*) = 1. If constant access to each simple node is possible, each *satisfy*(A) can be performed in $O(1)$ time.

We shall see now how to maintain the closure of the node *true* during the insertion of new clauses into a formula.

First of all, we extend the notion of status to compound nodes by defining

$$status(X) = \sum_{i \in X} status(i) \qquad \forall X \in N_c.$$

That is, the status of a compound node X is equal to the number of simple nodes in X which are not reachable from the node *true*; hence $status(X) = 0$ if and only if all the nodes in X are reachable from *true*. In order to implement the function status, we associate to each (simple and compound) node i an entry referred to as $STATUS[i]$.

At the beginning, when no clause has yet been inserted, our hypergraph consists of k simple nodes, and their *STATUS* entry is initialized by simply setting

$$STATUS[true] = 0,$$
$$STATUS[x] = 1 \qquad \forall x \in N\text{-}\{true\}.$$

When a new clause has to be introduced, according to Definition 2.1 we have to insert a new hyperarc from a properly defined source set to a given simple node in $G(H_A)$. If all the (dotted or full) arcs leaving a (simple or compound) node w are organized in an adjacency list named $L(w)$, the following algorithm maintains the closure of the node *true* while inserting a hyperarc from a source set X to a node Q:

```
procedure insert ($X$ : source_set, $Q$ : node);
var $w$ : compound_node;
begin
    if $|X| = 1$ then
        insert $Q$ into $L(X)$
    else
        begin
            create a new compound node $X$;
            for each simple node $i$ in $X$ do insert $X$ into $L(i)$;
            $L(X) := \{Q\}$;
            $STATUS[X] := \Sigma_{i \in X} STATUS[i]$
        end;
    if $STATUS[X] = 0$ then closure($Q$)
end;
```

The aim of the procedure *closure* is to update the *STATUS* entry of the nodes by means of a *true*-propagation mechanism:

```
procedure closure($Q$ : node);
var $w$ : node;
begin
    if $STATUS[Q] \neq 0$ then
        begin
            $STATUS[Q] := STATUS[Q] - 1$;
            if $STATUS[Q] = 0$ then
                for each $w$ in $L(Q)$ do closure($w$)
        end
end;
```

The correctness of the approach, that is, the fact that the *STATUS* entry correctly implements the *status* function, hinges on the following theorem.

Theorem 3.1. After any insert operation, the following proposition (referred to as status correctness) holds: a simple node i is reachable from the node true if and only if $STATUS[i] = 0$.

PROOF. *"If" part*: We prove that any time the following properties hold for the data structure.

(P1) If i is a simple node and $STATUS[i] = 0$, then there is a hyperpath from *true* to i.

(P2) If w is a compound node representing a source set X, then the number of nodes in X not reachable from *true* does not exceed $STATUS[w]$.

We proceed by induction on the number of (full or dotted) arcs examined during the execution of the procedure closure.

At the beginning (when the Horn formula is empty), properties (P1) and (P2) trivially hold, since only $STATUS[true]$ is initialized to 0 and no compound node exists. On the other hand, every compound node introduced trivially satisfies property (P2).

Let us assume now that properties (P1) and (P2) hold before examining an arc (x, y) during the execution of the procedure closure. Since (x, y) can be examined if and only if $STATUS[x]$ has been set to 0, then:

(i) If (x, y) is full, $STATUS[y]$ will be set to 0 and there will be a hyperpath from *true* to x (and hence also from *true* to y) for the inductive hypothesis. Thus, properties (P1) and (P2) still hold.

(ii) If (x, y) is dotted, then $STATUS[y]$ is decremented by 1 but is still greater than or equal to the number of simple nodes in y not reachable from *true*, since for the inductive hypothesis x was reachable from *true*. Thus, properties (P1) and (P2) again hold.

"Only if" part: We shall prove:

(P3) If there is a hyperpath from *true* to a simple node i, then $STATUS[i] = 0$ by induction on the number of clause insertions performed.

At the beginning (when the Horn formula is empty), no node is reachable from the node *true* except the node *true* itself. Since $STATUS[true]$ is initialized to 0, the base of the induction holds.

Assume now that property (P3) holds before inserting a clause

$$c = \neg P_1 \vee \neg P_2 \vee \cdots \vee \neg P_q \vee Q, \qquad q \geq 1$$

(i.e., a hyperarc from a source set $X = \{P_1, P_2, \ldots, P_q\}$ to Q). Note that there is no loss of generality, since every clause can be expressed as c, provided that also nodes *true* and *false* are taken into account. We have the following two possibilities:

(i) If $\Sigma_{i=1,\ldots,q} STATUS[P_i]$ is greater than 0 before the insertion of the clause c, then neither the nodes reachable from *true* nor the nodes whose *STATUS* is 0 can change after the insertion of the clause c.

(ii) Consider now the case where $\Sigma_{i=1,\dots,q} STATUS[P_i] = 0$ before the insertion of the clause c, and assume by contradiction that after inserting c there is a simple node y reachable from *true* but whose *STATUS* is greater than 0. Let us denote by T the hyperpath from *true* to y. Clearly T must include the hyperarc corresponding to the clause c, because otherwise the inductive hypothesis would be violated. Two cases may arise:

(a) y coincides with Q;
(b) y is different from Q.

Case (a) is not possible, because $STATUS[Q]$ is set to 0 at the first step of the procedure closure. In case (b) there must be a hyperarc from $\{x_1, x_2, \dots, x_p\}$ $(p \geq 1)$ and p hyperpaths from *true* to $x_1, x_2, \dots, x_p$, denoted respectively by $H_1, H_2, \dots, H_p$. $\Sigma_{i=1,\dots,q} STATUS[x_i]$ must be greater than 0; otherwise either $\Sigma_{i=1,\dots,q} STATUS[x_i]$ was equal to 0 before the insertion of the clause c (clearly contradicting the inductive hypothesis) or the nodes x_i with nonnull *STATUS* were forced to 0 after the insertion of c (and thus also y must have been forced to a null status). Hence, there must be a node x_i reachable from *true* by means of a hyperpath H_i but with *STATUS* greater than 0. Also in this case, H_i must include the hyperarc corresponding to the clause c (otherwise the inductive hypothesis would be violated). Since $\text{rank}(H_i) < \text{rank}(T)$, by repeating this argument we should eventually find that node Q itself should have *STATUS* greater than 0, which is clearly a contradiction as stated in case (a) above.

This completes the induction step and gives the theorem. □

We are now able to state the following lemma which characterizes the overall time complexity of our on-line algorithm and provides the basis for the subsequent results.

Lemma 3.1. The closure of the node true can be maintained in at most $O(n)$ time upon the insertion of any number of new clauses, where n is the total length of the formula.

Proof. A (full or dotted) arc (x, y) may be scanned immediately after the insertion of a new clause only if $STATUS[x]$ was greater than 0. Once scanned, $STATUS[x]$ is set to 0 and henceforth the arc (x, y) cannot be scanned again during subsequent calls of the procedure closure. Since the total number of arcs in the FD graph is equal to the length of the formula, the theorem is proved. □

The main result of the paper can now be stated, as the following theorem shows.

Theorem 3.2. Given a Horn formula A, there exists a data structure which allows one to check on line in $O(1)$ time whether a formula is satisfiable. The amortized cost of inserting a new clause is $O(q)$, where q is the length of the clause. The space required is $O(n)$.

Proof. Due to Theorem 2.1, our algorithm takes $O(1)$ time to answer each satisfiability question. As a consequence of Lemma 3.1, the total time involved in maintaining the data structure while new clauses are inserted is $O(n)$, where n is the length of the formula. Using the credit technique of Tarjan [17], it is immediate to see that the insertion of a clause of length q requires exactly q credits and therefore can be performed in $O(q)$ amortized time. □

This outperforms the best previously known algorithms for the same problem [6, 9], which can require even $O(n)$ time for testing the satisfiability of the formula each time a new clause is inserted. Lemma 3.1 can be easily generalized, as the following corollary shows.

Corollary 3.1. The closure of any node in a hypergraph corresponding to a Horn formula can be maintained in at most $O(n)$ time and space during the insertion of new clauses, where n is the total length of the formula.

Notice that from the point of view of the off-line computation the method that we have shown has the same efficiency of the one used in [6], but our data structure seems to be crucial for operating on line without loss of efficiency. Also, our data structure allows us to support other operations on line, as we shall see in the next section.

4. ON-LINE ALGORITHMS FOR MAINTAINING TRUTH VALUES AND IMPLICATIONS AMONG PROPOSITIONAL VARIABLES

In this section we extend the previous results to other operations which may be efficiently supported by the same data structure described above. Consider the case in which one would like to check on line whether any variable P can get a given truth value (*true* or *false*) in a satisfying interpretation of a Horn formula A. As an immediate consequence of Corollary 2.2, a variable P can get a truth value *true* [*false*] if and only if there is no hyperpath in H_A (the hypergraph corresponding to A) from $\{P, \mathit{true}\}$ to *false* [respectively, from *true* to P]. Hence, one can check whether a variable may have truth value *false* in constant time by using the data structure described in the previous section. In order to check that a variable may assume truth value *true*, we can use the same data structure provided that the definition of status is generalized as follows:

$$\mathit{status}_P(Q) = \begin{cases} 0 & \text{if there is a hyperpath from } \{P, \mathit{true}\} \text{ to } Q, \\ 1 & \text{otherwise} \end{cases}$$

for each $P, Q \in N$ (i.e., simple nodes);

$$\mathit{status}_P(X) = \sum_{i \in X} \mathit{status}_P(i)$$

for each $P \in N$ (i.e., simple node) and $X \in N_c$ (i.e., compound node).

With this generalization, a propositional variable P can get a truth value *true* in a satisfying interpretation of a Horn formula A if and only if $\mathit{status}_P(\mathit{false}) = 1$. By generalizing also the *STATUS* entries of our data structure, this test can be accomplished in constant time. Furthermore, the truth assignment

$$v(Q) = \begin{cases} \mathit{true} & \text{if and only if } \mathit{status}_P(Q) = 0, \\ \mathit{false} & \text{otherwise} \end{cases}$$

is the least truth assignment in the boolean algebra $\{\mathit{false},\mathit{true}\}^k$ satisfying A in which P gets truth value *true*. As a consequence, the generalized data structure allows one to find a truth assignment satisfying A in which a given propositional variable gets truth value *true* in $O(k)$ time.

The total time involved in maintaining the data structure during the insertion of new clauses is $O(kn)$, since the closure of at most k nodes has to be maintained on line (see Corollary 3.1).

The space complexity of the generalized data structure can be analyzed as follows. The *STATUS* entries need now $O(|N|(|N| + |N_c|))$ storage. Furthermore, the hypergraph corresponding to A can be maintained in $O(n)$ space. Since $|N| = k$, $|N_c| = m$, and $n \leq mk$, the overall space is $O((m+k)k)$. The above argument leads to the following theorem.

Theorem 4.1. Given a Horn formula A containing at most k different propositional variables and m clauses, there exists a data structure which allows one

(i) *to check on line in* $O(1)$ *time whether the formula is satisfiable*;

(ii) *to check on line in* $O(1)$ *time whether a propositional variable can get truth value true (or false) in a satisfying interpretation of A*;

(iii) *to find in* $O(k)$ *time the least truth assignment in the boolean algebra* $\{\mathit{false},\mathit{true}\}^k$ *satisfying A in which a given propositional variable gets truth value true (or false).*

The total time involved in maintaining the data structure while new clauses are inserted is $O(kn)$, *where n is the length of the formula. The space required is* $O((m+k)k)$.

Note that, if no additional structure were maintained, performing every time from scratch the operations in (i), (ii), and (iii) could require even $O(n)$ time, where n is the total length of the formula. If we now compare this bound with the performance of the data structure, we may observe that over a sequence of any number of insertions of clauses and q tests of type (i), (ii), and (iii) our data structure requires an overall time of $O(k(n+q))$, which compares favorably with $O(qn)$ when $q > k$ [$k < n$, and q might be as large as mk, since after any insertion, k type-(ii) tests could be performed].

Example 4.1. Consider the following Horn formula:

$$B \wedge E \wedge (\neg A \vee \neg B \vee \neg C \vee D) \wedge (\neg A \vee H) \wedge (\neg D \vee \neg E \vee F)$$
$$\wedge (\neg D \vee \neg F \vee G) \wedge (\neg H \vee I) \wedge (\neg G \vee \neg I) \wedge (\neg F).$$

In Figure 5 is shown its corresponding hypergraph. The variable C can get a truth value *true* in a satisfying interpretation of the formula, since there is no hyperpath

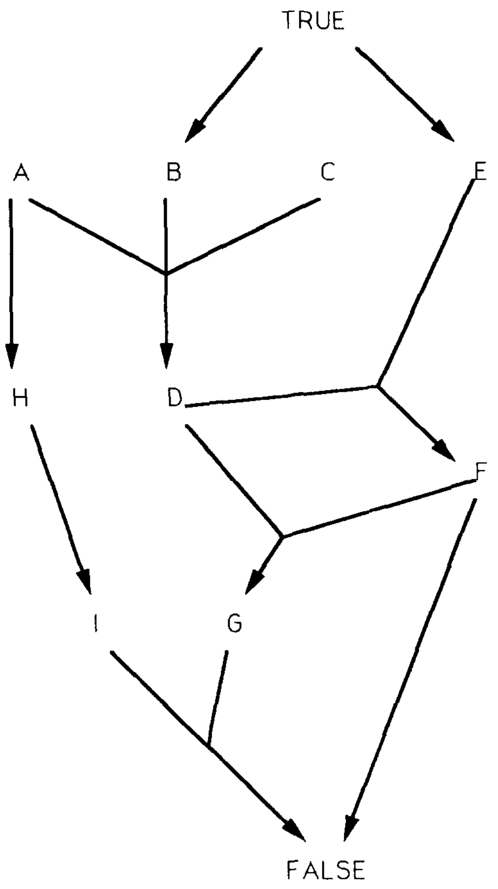

FIGURE 5. The hypergraph corresponding to the formula given in Example 4.1.

from $\{C, true\}$ to *false*. In fact

$$v(P) = \begin{cases} true & \text{if } P = B \text{ or } P = C \text{ or } P = E, \\ false & \text{otherwise} \end{cases}$$

satisfies the formula. On the other hand, the variable E cannot assume truth value *false* in any satisfying interpretation of the formula, since there is a hyperpath from *true* to E.

Operation (ii) in Theorem 4.1 corresponds to checking whether an implication between two variables P and Q in the formula holds, i.e. to checking whether a variable Q must have truth value *true* once another variable P has been given truth value *true*. Since this may be seen as a first step toward maintaining information about implications or deductions within a given set of rules, we may extend it to the set of variables which appear negated in a clause (in the following referred to as a *premise* of a clause). Hence, we consider the situation in which, besides checking the satisfiability of a formula, we would like to maintain information about the consequences of assuming that all facts which constitute the premise of a particular clause are assumed to be true. This gives rise to the following theorem, whose proof recalls completely Theorem 4.1 and thus has been omitted.

Theorem 4.2. Given a Horn formula A containing at most k different propositional variables and m clauses, there exists a data structure which allows one

(i) *to check on line in $O(1)$ time whether the formula is satisfiable;*

(ii) *to check on line in $O(1)$ time whether either any propositional variable or all the variables of a given premise can get truth value true (or false) in a satisfying interpretation of A;*

(iii) *to find in $O(k)$ time the least truth assignment in the boolean algebra $\{false, true\}^k$ satisfying A in which either a given propositional variable or all the variables of a given premise get truth value true (or false).*

The total time involved in maintaining the data structure while new clauses are inserted is $O((k+m)n)$, where n is the length of the formula. The space required is $O((m+k)^2)$.

5. ON-LINE SATISFIABILITY OF A CLASS OF NON-HORN FORMULAE

Yamasaki and Doshita studied in [19] the satisfiability problem for a class S_0 of propositional formulae in CNF which properly contains all Horn formulae. A formula S is in such a class if there is an ordering of the clauses $c_1, c_2, \ldots, c_m$ such that

(1) $c_i = H_i \vee P_i$, where each H_i is a Horn clause and each P_i is a disjunction of positive literals;

(2) $P_{i+1} \subset P_i$, $i = 1, 2, \ldots, m$.

Motivations for studying such a class arise from considering classes which include Horn formulae and whose satisfiability problem remains polynomial. In [19] an $O(n^3)$ algorithm for this satisfiability problem has been given, where n is the length of the input. Recently, in [1] Arvind and Biswas proposed a more efficient algorithm which requires $O(kn)$ time and $O(mk)$ space, where k is the number of propositional variables and m the number of clauses in the formula. Actually, an extra cost of $O(m \log m)$ time is required for sorting the clauses (if they are not sorted in a preprocessing step).

In this section we shall provide an on-line Arvind-Biswas algorithm for the satisfiability problem in S_0, based on the same data structures considered in the previous sections. Each question about satisfiability is answered in constant time, while the total time involved in maintaining the formula during the insertion of new clauses is $O(kn + m \log m)$, as in the best known off-line algorithms [1]. The space required by our algorithm is $O(n)$. Since $n < km$, this result not only reduces the on-line complexity of this problem, but also optimally improves the space complexity given in [1].

Assume now that we are given a formula $S = c_1 \wedge c_2 \wedge \cdots \wedge c_m$ in S_0, where $c_i = H_i \vee P_i$, whose satisfiability must be analyzed. It can be proved (see [1] for further details) that S is satisfiable if and only if one of the following conditions holds:

(1) *r-satisfiability*: There exists a propositional variable $p \in P_r$, $r \geq 1$, which gets truth value *true* in a satisfying interpretation of the Horn formula $A_r = H_{r+1} \wedge H_{r+2} \wedge \cdots \wedge H_m$. p will be referred in the sequel as the *witness* of the r-satisfiability.

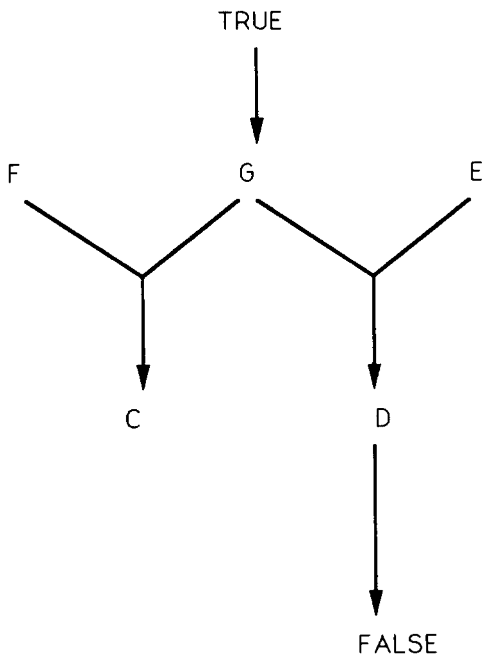

FIGURE 6. The hypergraph corresponding to a Horn part of the formula given in Example 5.1.

(2) 0-*satisfiability*: The Horn formula $A_0 = H_1 \wedge H_2 \wedge \cdots \wedge H_m$ is satisfiable (without witnesses).

Example 5.1. The formula consisting of the clauses

c_1:	$(\neg A \vee \neg B \vee \neg C) \vee (E \vee F \vee G \vee H)$
c_2:	$(\neg H \vee B) \vee (E \vee F)$
c_3:	$(\neg G \vee \neg F \vee C) \vee (E)$
c_4:	$(\neg E \vee \neg G \vee D)$
c_5:	(G)
c_6:	$(\neg D)$

is in S_0. In Figure 6 is shown the hypergraph corresponding to $H_3 \wedge H_4 \wedge H_5 \wedge H_6$. Since there is a hyperpath from $\{true, E\}$ to *false*, owing to Corollary 2.2. E cannot get truth value *true* in a satisfying interpretation of $H_4 \wedge H_5 \wedge H_6$. As a consequence, E cannot be a witness for the satisfiability of S. However, a 2-satisfiability witness can be found in F.

Consider now the insertion of a new clause $c = H \vee P$ into S, which gives rise to a new formula $S' = S \wedge c$ still in S_0.

If S was unsatisfiable, S' will also be unsatisfiable, and thus the clause c does not add any further information to our problem.

In the opposite case (S was satisfiable), we maintain the greatest integer r in $[0, m]$ such that S is r-satisfiable together with the witness p of this r-satisfiability (if $r \geq 1$). First of all, the position s in which the clause c must be inserted, given by $P_s \subset P \subset P_{s+1}$, must be individuated. Furthermore, three cases must be discriminated depending on s and r.

If $s < r$, then S' is still satisfiable, since the witness p also makes the clause c satisfied. Moreover, the Horn formula $A_r = H_{r+1} \wedge \cdots \wedge H_m$ has not been affected by c.

If $s > r$, we must check whether the witness p gets still truth value *true* in a satisfying interpretation of the modified Horn formula

$$A_r = H_{r+1} \wedge \cdots \wedge H_s \wedge H \wedge H_{s+1} \wedge \cdots \wedge H_m.$$

In the affirmative case, S' is still r-satisfiable. Otherwise, the other propositional variables not yet examined in $P_r, P_{r-1}, \ldots, P_1$ must be scanned in order to get either a new witness for l-satisfiability $(0 \le l \le r)$ or an unsatisfiability answer in case of failure.

Finally, if $s = r$, two cases are possible. If $p \in P$, then S' is still satisfiable. In the other case, the variables not yet examined in $P, P_r, \ldots, P_1$ must be scanned in order to get either a new witness for an l-satisfiability $(0 \le l \le r+1)$ or an unsatisfiability answer.

We now give a brief description of the data structure used for dealing with a formula S in S_0. Next, an algorithm for the on-line satisfiability of formulae in S_0 will be presented.

If S is r-satisfiable $(r \ge 0)$, we maintain the hypergraph $\boldsymbol{H}$ corresponding to the Horn formula $A_r = H_{r+1} \wedge \cdots \wedge H_m$ subject to the operations *insert* (insertion of a Horn clause into a Horn formula) and *satisfy* (check for deciding whether the Horn formula so far achieved is satisfiable) defined in the previous sections. Furthermore, the clauses c_i $(1 \le i \le r)$ are maintained in a heap, where each item points to a list containing the subclause H_i and to a list containing the subclause P_i. If $r \ge 1$, the witness of the r-satisfiability is the *head* (i.e., the first item) of the list P_r. On the other hand, if S is unsatisfiable, only the hypergraph corresponding to the Horn formula $A_0 = H_1 \wedge \cdots \wedge H_m$ is maintained.

When a variable $p \in P_r$ is discarded as a witness due to the insertion of a new clause in position $s > r$, p could be eliminated from the lists P_i $(1 \le i \le r)$, since it will no longer be a witness of satisfiability in the future. In order to save time during these operations, we simply mark the variable p as dead, by means of a suitable marking system. While looking for a new witness in a list P_i, the scanned variables which are marked as dead will be removed. Furthermore, the maximum integer r which gives r-satisfiability is maintained. If the formula is unsatisfiable, r is undefined.

Having filled in all the details of the data structure, we show how to insert a new clause into a formula in S_0:

```
**procedure** $S_0$_*insert*($c = H \vee P$ : *clause*);
**begin**
    "locate the position $s$ in which the clause $c$ has to be inserted";
    **case** $s$ **of**
        $s > r$:  **begin**
                      *insert*($H$, $\boldsymbol{H}$);
                      **if not** *satisfy*($\boldsymbol{H}$) **then**
                          **begin**
                              "mark *head*($P_r$) as dead";
                              new_witness
                          **end**
                  **end**;
        $s < r$:  "insert the clause $c = H \vee P$ into the heap with key $|P|$";
        $s = r$:  begin
                     "insert the clause $c = H \vee P$ into the heap with key $|P|$";
                     $r := r + 1$;
                     if $p \in P$ then
                         "move $p$ at the head of the list $P_r$"
                     else
                         new_witness
                  end
    endcase
end;
```

The procedure *new_witness*, called when the current witness of r-satisfiability fails, tries to find a new witness which can ensure the satisfiability again. The idea is to scan the lists P_i, $i = r, r-1, \ldots, 1$, from left to right in a bottom-up fashion (i.e., starting with P_r). If a nondead variable $p \in P_k$, $k \leq r$, is encountered, then it must be verified whether p can get a truth value *true* in a satisfying interpretation of the Horn formula $A_k = H_{k+1} \wedge \cdots \wedge H_m$ in order to witness the satisfiability of the whole formula.

According to Corollary 2.2, this can happen if and only if there is no hyperpath from $\{true, p\}$ to *false* in the hypergraph $\boldsymbol{H}_k$ corresponding to the Horn formula A_k. This task can be accomplished by adding the Horn clauses $H_{k+1}, \ldots, H_r$ to the hypergraph $\boldsymbol{H}_r$ corresponding to the Horn formula $A_r = H_{r+1} \wedge \cdots \wedge H_m$. During these operations, the closure of the node *true* is maintained as described in the previous sections. Furthermore, the closure of $\{true, p\}$ is computed and maintained on line as long as p is useful as a witness.

The above argument gives rise to the following procedure:

```
procedure new_witness;
begin
    if $P_r \neq \emptyset$ then
        begin
            $p := head(P_r)$;
            if "$p$ is marked dead" then
                begin
                    remove $head(P_r)$;
                    new_witness
                end
            else
                begin
                    "compute the closure of $\{true, p\}$ in $\boldsymbol{H}$";
                    if not satisfy($\boldsymbol{H}$) then
                        begin
                            "mark $p$ as dead";
                            new_witness
                        end
                end
        end
    else
        begin
            if $r > 0$ then
```

```
                begin
                    $\langle H_r, P_r \rangle := deletemin(heap)$;
        {return the clause $H_r \vee P_r$ whose $P_r$ is of minimum size}
                        $insert(H_r, H)$;
                        $r := r - 1$;
                        new_witness
                end
        end
end;
```

The correctness of the algorithm is the consequence of both Corollary 2.2 and Theorem 3.1 together with the fact that it is an adaptation of the algorithm proposed in [1]. With regard to its complexity, we prove the following result.

Theorem 5.1. The algorithm for the on-line satisfiability of formulae in S_0 requires a total of $O(kn + m \log m)$ time and $O(n)$ space, where k is the number of distinct propositional variables, m the total number of clauses to deal with, and n the length of the formula.

Proof. The data structure consists of one heap, one hypergraph, and at most $2m$ lists (H_i and P_i).

While maintaining the heap, we are interested both in removing the clause of minimum key in the heap, and in inserting a clause with respect to its key. Since, once removed, a clause can never be reinserted into the heap, the total time involved in maintaining the heap is $O(m \log m)$.

As far as the hypergraph is concerned, we have to insert new hyperarcs (i.e., new Horn clauses) while maintaining information about hyperpaths. In more detail, given a witness p, Corollary 2.2 suggests maintaining the closure of $\{true,p\}$ while new hyperarcs are inserted. This task can be accomplished by introducing one dummy node w and two dummy hyperarcs $(w \rightarrow true)$ and $(w \rightarrow p)$ and by maintaining (with the algorithm previously described) the closure of w instead of $\{true,p\}$. Corollary 3.1 will then assure that maintaining information about the satisfiability with a given witness will cost $O(n)$ time and space. Since, once eliminated, a witness cannot any longer be taken into account, this implies a total of $O(|P_1|n) \approx O(kn)$ worst-case time for maintaining the hypergraph.

With regard to the lists P_i and H_i, their scanning will require a total of $O(n)$ time. If the marks of the witnesses considered are organized as a balanced search tree [5] containing as items the propositional variables with the information whether they are marked dead or not, the scanning process during the subsequent calls of the procedure *new_witness* can be performed in a total of $O(n \log|P_1|) \approx O(n \log k)$ time, while the deletion of the examined witnesses from the lists yields to a total of $O(n)$ time.

As a consequence, the total time involved in maintaining the data structure is $O(nk + m \log m)$.

The space required by the heap, the hypergraph, the lists, and the marking search tree is clearly $O(n)$. □

Clearly, the results stated in Theorems 4.1 and 4.2 might also be extended to the new class of non-Horn formulae within substantially the same time and space bounds.

6. CONCLUSION

The development of applications of logic programming and the use of rule based systems in knowledge representation has brought the need for data structures and algorithms for efficiently maintaining sets of logical formulae. In this paper, the problem of performing on-line sequences of operations such as insertions of new clauses and tests for satisfiability has been investigated.

Due to the time complexity and intractability of the general case [2, 13], only Horn formulae have been considered, and the results have been extended to a particular class of non-Horn formulae already considered in the literature. For both classes it is shown how the given operations may be performed on line efficiently. We have presented techniques which significantly outperform previously known solutions of the same problems. Also, new on-line operations such as checking the consistency of a truth assignment and deriving implications of given hypotheses have been considered and their overall cost analyzed.

In order to satisfy the needs of a real environment for the design of rule based systems [2, 13], the time and space complexity of other operations should be analyzed, such as deletion of clauses, backtracking, etc. Research in these directions is currently being developed.

We would like to thank the referees for their helpful comments and valuable suggestions, which improved the presentation of the paper.

REFERENCES

1. Arvind, V. and Biswas, S., An $O(n^2)$ Algorithm for the Satisfiability Problem of a Subset of Propositional Sentences in CNF that Includes All Horn Sentences, *Inform. Process. Lett.* 24:67–69 (1987).
2. Apt, K. R. and Pugin, J. M., Management of Stratified Data Bases, Technical Report TR-87-41, Dept. of Computer Science, Univ. of Texas at Austin, 1987.
3. Ausiello, G., D'Atri, A., and Saccà, D., Graph Algorithms for Functional Dependency Manipulation, *J. Assoc. Comput. Mach.* 30:752–766 (1983).
4. Ausiello, G., D'Atri, A., and Saccà, D., Minimal Representation of Directed Hypergraphs, *SIAM J. Comput.* 15:418–431 (1986).
5. Aho, A. V., Hopcroft, J. E., and Ullman, J. D., *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass., 1974.
6. Dowling, W. F. and Gallier, J. H., Linear-Time Algorithms for Testing the Satisfiability of Propositional Horn Formulae, *J. Logic Programming* 3:267–284 (1984).
7. Even, S. and Shiloach, Y., An On-Line Edge Deletion Problem, *J. Assoc. Comput. Mach.* 28:1–4 (1981).
8. Frederickson, G. N., Data Structures for On-Line Updating of Minimum Spanning Trees with Applications, *SIAM J. Comput.* 14:781–798 (1985).
9. Itai, A. and Makowsky, J., Unification As a Complexity Measure for Logic Programming, *J. Logic Programming* 4:105–117 (1987).
10. Italiano, G. F., Amortized Efficiency of a Path Retrieval Data Structure, *Theoret. Comput. Sci.* 48:273–281 (1986).
11. Jones, N. D. and Laaser, W. T., Complete Problems for Deterministic Polynomial Time, *Theoret. Comput. Sci.* 3:107–117 (1977).
12. Kowalski, R., *Logic for Problem Solving*, Elsevier, New York, 1979.

13. Lassez, C., McAloon, K., and Port, G., Stratification and Knowledge Base Management, in: *Proceedings of the 4th International Conference on Logic Programming*, Melbourne, 1987, pp. 136–151.

14. Nilsson, N. J., *Problem Solving Methods in Artificial Intelligence*, McGraw-Hill, New York, 1971.

15. Petri, C. A., Communication with Automata, Tech. Rep. RADC-TR-65377, Vol. 1, Griffith Air Force Base, New York, 1966.

16. Sleator, D. D. and Tarjan, R. E., A Data Structure for Dynamic Trees, *J. Comput. System Sci.* 26:362–391 (1983).

17. Tarjan, R. E., Amortized Computational Complexity, *SIAM J. Algebraic Discrete Methods* 6:306–318 (1985).

18. Ullman, J. D., *Principles of Database Systems*, Computer Science Press, Rockville, Md., 1982.

19. Yamasaki, S. and Doshita, S., The Satisfiability Problem for a Class Consisting of Horn Sentences and Some Non-Horn Sentences in Propositional Logic, *Inform. and Control* 59:1–12 (1983).