A NOTE ON DOWLING AND GALLIER'S TOP-DOWN ALGORITHM FOR PROPOSITIONAL HORN SATISFIABILITY

MARIA GRAZIA SCUTELLÀ

▷ In "Linear time algorithms for testing the satisfiability of propositional Horn formulae" (*J. Logic Programming*, 1984), Dowling and Gallier have presented two linear-time algorithms for checking the satisfiability of a propositional Horn formula. In this note we show that one of these algorithm, the top-down one, may under particular circumstances not give the correct answer, and we propose a correct version of the algorithm which also runs in linear time. ◁

1. INTRODUCTION

In [1] two linear algorithms for checking the satisfiability of a propositional Horn formula S are presented. These algorithms visit a graph G_S which describes the implicational structure of the formula S. The nodes of the graph are the distinct propositional letters in S plus two special nodes, one for TRUE (the proposition which is always true) and one for FALSE (the proposition which is always false). The edges of the graph are labeled with the basic Horn formulae of S: a *basic Horn formula* is a disjunction of propositional letters with at most one positive letter. They are defined as follows:

(1) If the ith basic Horn formula in S is a positive letter P, in G_S there is an edge, from P to TRUE, labeled i.

(2) If the ith basic Horn formula in S is of the form $\neg P_1 \vee \cdots \vee \neg P_p$, in G_S there are p edges, from FALSE to $P_1, \ldots, P_p$, labeled i.

Address correspondence to M. G. Scutellà, Dipartimento di Informatica, Università di Pisa, 56100 Pisa, Italy.
Received February 1988; accepted September 1988.

(3) If the ith basic Horn formula in S is of the form $\neg P_1 \vee \cdots \vee \neg P_p \vee P$, in G_S there are p edges, from P to $P_1, \ldots, P_p$, labeled i.

The fundamental property of the graph is that S is unsatisfiable if and only if there is in G_S a path of a certain kind, called *pebbling*, between the nodes TRUE and FALSE. The first algorithm finds a pebbling, if it exists, in a bottom-up fashion, by proceeding from the node TRUE to the node FALSE. Such an algorithm, which is based on a breadth-first visit of the graph, is simple, elegant, and very efficient too. Gallo and Urbani [2], who use it as the core of an enumerative algorithm for the general propositional satisfiability problem, present some experimental evidence of its computational efficiency.

The second algorithm finds a pebbling in a top-down fashion, by proceeding backward from the node FALSE to the node TRUE. In this note we show that the top-down algorithm as described by Dowling and Gallier may under particular circumstances not give the correct answer, and we propose a correct version of the algorithm running in linear time.

2. ANALYSIS OF DOWLING AND GALLIER'S TOP-DOWN ALGORITHM

The Dowling-Gallier top-down algorithm visits the graph G_S to show the unsatisfiability of S. S is unsatisfiable if and only if in S there is a basic Horn formula $\neg P_1 \vee \cdots \vee \neg P_p$ such that all P_i, $i = 1, \ldots, p$, must be true: a propositional letter P which appears in S *must be true* if and only if in S there is a basic Horn formula of the form P, or of the form $\neg P_1 \vee \cdots \vee \neg P_{p'} \vee P$ where $P_1, \ldots, P_{p'}$ must be true (i.e. P must be true if and only if P is bound to the value *true* in each assignment satisfying S). For example, $S = \{\neg P_1 \vee \neg P_2, P_1, P_2\}$ is unsatisfiable, because P_1 and P_2 must be true.

By considering the graph G_S, we see that a node P must be true if and only if there is a label i such that the only edge labeled i has P as tail and TRUE as head, or such that all the edges labeled i have P as tail and nodes which must be true as heads (trivially the node TRUE must be true). S is unsatisfiable if and only if, for some label i, all the edges labeled i have FALSE as tail and nodes which must be true as heads (i.e. if and only if FALSE must be true).

Therefore the top-down algorithm is implemented by Dowling and Gallier as a recursive procedure, TRAVERSE, which, given a node P, finds recursively whether P must be true, by beginning from the node FALSE. The difficulty is that the graph G_S may have cycles. For instance, if $S = \{\neg P_1, \neg P_2 \vee P_1, \neg P_1 \vee \neg P_3 \vee P_2, P_3\}$, then G_S has one cycle (see Figure 1).

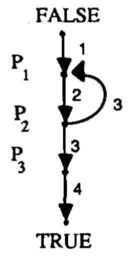

FIGURE 1.

It follows that it is necessary to use a marking technique to prevent the procedure from looping. In their paper Dowling and Gallier propose the following strategy. For checking whether a node P must be true, only the immediate successors of P connected to it by an unvisited edge, or, if no such node exists, the immediate successors of P having some unvisited outgoing edge, can be recursively visited by TRAVERSE. To implement this strategy, a field VISITED is associated to each edge, and a field MARKED to each node of the graph. MARKED is a counter holding the number of unvisited edges outgoing from a node. If P must be true, a field VAL(P) is first set to *true*, and then also a field COMPUTED(P) is set to *true*, to avoiding recomputing VAL(P). Anyway COMPUTED(P) is set to *true* when no immediate successors of P can be visited, i.e. when all the edges outgoing from P and from all the immediate successors of P have been visited: in this case VAL(P) can no longer be computed.

Initially, VAL and COMPUTED are set to *true* for the nodes corresponding to basic Horn clauses consisting of a single positive literal and for TRUE, and are set to *false* for the other nodes; VISITED and MARKED are set respectively to *false* and to the number of edges outgoing from each node. The initialization is performed by the procedure BUILDGRAPH.

Dowling and Gallier's algorithm can be stated in a formal way as follows, where for the sake of simplicity we do not include declarations of data and variables types:

```
Procedure TOP-DOWN($S$)
  begin
  BUILDGRAPH($G_S$);
  if no positive unit clause exists in $S$ then print '$S$ is satisfiable'
  else
    begin
    TRAVERSE(FALSE);
    if VAL(FALSE)
      then print '$S$ is unsatisfiable'
      else print '$S$ is satisfiable' and ⟨compute a truth assignment for $S$⟩
    end
  end.

Procedure TRAVERSE($P$)
  if $\neg$COMPUTED($P$) then
    begin
    if VAL($P$) then COMPUTED($P$) := true else
      begin
      TAGSET := $\{i : i$ labels some edge outgoing from $P\}$;
      for each $i$ in TAGSET and $\neg$VAL($P$) do
        begin
        ARC := $\{e : e$ is an edge outgoing from $P$ labeled $i\}$;
        NODE := $\{P' : P'$ is the head of an edge in ARC$\}$;
        for each $e$ in ARC do
          begin
          $Q$ := head($e$);
          if $\neg$VISITED($e$) then
                      begin
                      VISITED($e$) := true;
```

```
                    MARKED($P$) := MARKED($P$) − 1;
                    TRAVERSE($Q$)
                    end
                  else
                    if MARKED($P$) = 0 and MARKED($Q$) ≠ 0
                      then TRAVERSE($Q$)
      end;
    if ¬COMPUTED($P$) then
      if VAL($P'$) = true for each $P'$ in NODE then VAL($P$) := true;
    end;
  COMPUTED($P$) := true;
  end;
end.
```

Unfortunately the above algorithm may under particular circumstances not give the correct answer. Let us consider for instance the following propositional Horn formula:

$$S = \{\neg P_1 \vee \neg P_2, P_1 \vee \neg P_3, P_3 \vee \neg P_4,$$

$$P_4 \vee \neg P_5, P_4 \vee \neg P_6, P_6, P_5 \vee \neg P_1, P_2 \vee \neg P_5\}.$$

It is trivial to verify that S is unsatisfiable. The graph G_S corresponding to S is shown in Figure 2.

Let us suppose that the algorithm calls, in order, TRAVERSE(FALSE), TRAVERSE(P_1), TRAVERSE(P_3), TRAVERSE(P_4), TRAVERSE(P_5), and TRAVERSE(P_1). The last call to P_1 sets COMPUTED(P_1) to *true*, because for all the edges outgoing from P_1 and from the only successor of P_1 (i.e. P_3) the VISITED field is *true*. Obviously VAL(P_1) is still *false*. When in backtracking the algorithm returns to P_5, COMPUTED(P_5) is set to *true* for the same reason, whereas VAL(P_5) is always *false*. Subsequently the algorithm backtracks to P_4 and calls TRAVERSE(P_6): VAL(P_6) is *true*, and therefore VAL(P_4), VAL(P_3), and VAL(P_1) are set to *true*. At this point the algorithm calls TRAVERSE(P_2), which calls TRAVERSE(P_5). But TRAVERSE(P_5) returns immediately because COMPUTED(P_5) is *true*. It follows that VAL(P_2) remains *false* because VAL(P_5) is *false*, and therefore VAL(FALSE) remains *false*. The Dowling-Gallier top-down algorithm decides that S is satisfiable, whereas in fact it is unsatisfiable.

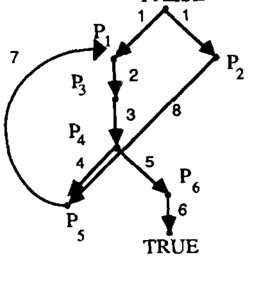

FIGURE 2.

3. MODIFICATION OF DOWLING AND GALLIER'S TOP-DOWN ALGORITHM

In the following we propose a modification of Dowling and Gallier's top-down algorithm which correctly decides whether a propositional Horn formula S is satisfiable. The most important difference between the two algorithms is their behavior in the presence of cycles.

Four boolean fields, initialized by the procedure BUILDGRAPH, are associated to each node P in G_S:

(1) VAL(P) is *true* if the algorithm verifies that P must be true, as in Dowling and Gallier's algorithm [initially VAL(P) is *true* for $P =$ TRUE and for each node P corresponding to basic Horn clauses consisting of a single positive letter, and it is *false* for the other nodes].

(2) COMPUTED(P) is *true* if VAL(P) has been computed top-down, as in Dowling and Gallier's algorithm [initially COMPUTED(P) is *true* for $P =$ TRUE and for each node P corresponding to basic Horn clauses consisting of a single positive letter, and it is *false* for the other nodes].

(3) SUSP(P) is *true* if the top-down evaluation of VAL(P) has begun, but it is not finished [initially SUSP(P) is *false* for each node P].

(4) LOOP(P) is *true* if a cycle is found in the top-down evaluation of VAL(P) [initially LOOP(P) is *false* for each node P].

The algorithm is implemented as a recursive procedure, EVALUE, which, given a node P, finds recursively whether P must be true [i.e. whether VAL(P) is true], by beginning from FALSE. Obviously S is unsatisfiable if and only if VAL(FALSE) is true.

Given a node P, the procedure EVALUE sets SUSP(P) to *true* and finds recursively whether for some i labeling edges outgoing from P, all the nodes $P_1, \dots, P_p$ which are heads of the edges labeled i must be true. In this case VAL(P) is set to *true*, else VAL(P) remains false. In both cases COMPUTED(P) is set to *true* and SUSP(P) is set again to *false*. To find VAL(P_1), ..., VAL(P_p), EVALUE calls a function MIN which has as input the set NODE $= \{P_1, \dots, P_p\}$. To find VAL(j) for each j in NODE, MIN has three possibilities:

(1) COMPUTED(j) is *true*, i.e. the algorithm has evaluated VAL(j) top-down. In this case it is sufficient to read VAL(j).

(2) COMPUTED(j) is *false* and SUSP(j) is *false*, i.e. the algorithm has never evaluated VAL(j) top-down. In this case the function MIN calls the procedure EVALUE recursively, on input j.

(3) SUSP(j) is true, i.e. VAL(j) is demanded inside the top-down evaluation cycle of VAL(j). In this case, to prevent the procedure from looping, MIN must consider VAL(j) as false. MIN must also signal the presence of the cycle, and the field LOOP(j) is set to *true*. When, in the backtracking phase, the algorithm returns to node j, if it changes the value of VAL(j) from *false* to *true*, then it is necessary to visit again the subgraph with root j, i.e. the subgraph of G_S that the algorithm has traversed in the top-down visit in order to find the correct value of VAL(j). In fact, this subgraph may have

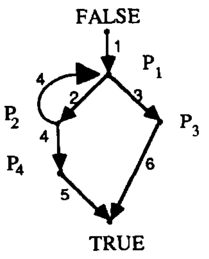

FIGURE 3.

some node P which has been evaluated to *false* [i.e. VAL(P) is false] by considering VAL(j) as *false*, whereas now, as a consequence of the new value of VAL(j), it should be evaluated to *true*. Such a visit can be performed in a bottom-up fashion on the subgraph with root j, by beginning from j; of course, only nodes P with VAL(P) $=$ *false* must be considered, since the value of the nodes already set to *true* cannot be further modified.

We observe that this bottom-up visit can be implemented as in [1], and therefore it is linear. We also observe that each edge in G_S can be bottom-up visited one time at most.

It is easy to see that the bottom-up visits do not change the top-down nature of the algorithm. In fact the subgraph of G_S visited by the algorithm is determined in a top-down fashion, and the bottom-up visits have to be considered only as an instrument to correctly propagate the value *true* in that subgraph whenever needed, i.e. in the presence of cycles.

Example. Let us consider $S = \{\neg P_1, P_1 \vee \neg P_2, P_1 \vee \neg P_3, P_2 \vee \neg P_1 \vee \neg P_4, P_4, P_3\}$. The graph G_S corresponding to S is shown in Figure 3.

Let us suppose that the algorithm calls, in order, EVALUE(FALSE), EVALUE(P_1) and EVALUE(P_2). To find VAL(P_2) it is necessary to find VAL(P_1), while VAL(P_2) is demanded to find VAL(P_1). Therefore to find VAL(P_2) the algorithm considers VAL(P_1) as *false* and sets LOOP(P_1) to *true* (see Figure 4).

When in backtracking the algorithm returns to P_1, it verifies that VAL(P_3) is *true*, and therefore VAL(P_1) is set to *true*. Then the algorithm visits in a bottom-up fashion the subgraph with root P_1 (dashed in Figure 5), by beginning from P_1. In this way VAL(P_2) is set to *true*.

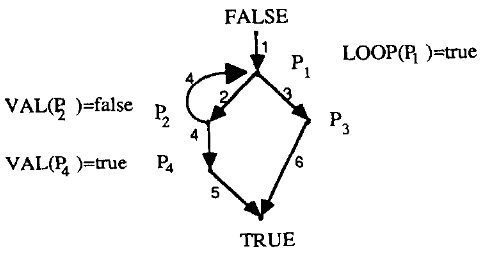

FIGURE 4.

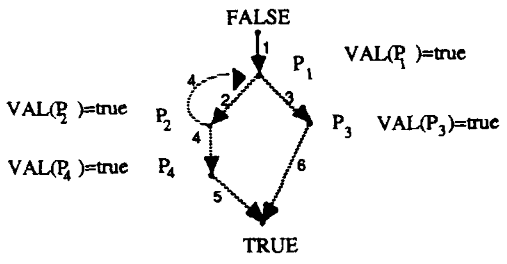

FIGURE 5.

Now we present the algorithm in a more formal way:

```
Procedure MODIFIED TOP-DOWN($S$)
  begin
  BUILDGRAPH($G_S$);
  TEMP := EVALUE(FALSE);
  if TEMP
    then print '$S$ is unsatisfiable'
    else print '$S$ is satisfiable'
  end.

Function EVALUE($P$)
  begin
  SUSP($P$) := true;
  TAGSET := $\{i: i$ labels some edge outgoing from $P\}$;
  for each $i$ in TAGSET and $\neg$VAL($P$) do
    begin
    ARC := $\{e: e$ is an edge outgoing from $P$ labeled $i\}$;
    NODE := $\{P': P'$ is the head of an edge in ARC$\}$;
    TEMP := MIN(NODE);
    if TEMP then
      begin
      VAL($P$) := true;
      if LOOP($P$) then
        begin
        ⟨Find the subgraph with root $P$, i.e. $G_P$⟩
        BOTTOMUP-VISIT($G_P$);
        end;
      end;
    end;
  SUSP($P$) := LOOP($P$) := false;
  COMPUTED($P$) := true;
  return VAL($P$)
  end.

Function MIN(NODE)
  begin
```

```
*NUMBER is the number of nodes in NODE with VAL equal to false after the
evaluation*
NUMBER := 0;
for each j in NODE do
  begin
  if COMPUTED(j) then
    begin
    if ¬VAL(j) then NUMBER := NUMBER + 1;
    end
                  else
                  if ¬SUSP(j) then
                    begin
                    TEMP := EVALUE(j);
                    if ¬TEMP then NUMBER := NUMBER + 1
                    end
                          else
                            begin
                            LOOP(j) := true;
                            NUMBER := NUMBER + 1
                            end
  end;
if NUMBER > 0 then return false else return true
end.
```

Theorem. MODIFIED TOP-DOWN *correctly decides whether S is unsatisfiable.*

PROOF. If G_S is acyclic, the algorithm is obviously correct. Therefore let us suppose that G_S has cycles. Let $N = \{\text{FALSE}, P_2, \dots, P_n\}$ be the set of nodes visited by the algorithm, and let N' be the set of nodes of N corresponding to clauses consisting of a single positive letter.

We observe that the values VAL(FALSE), ..., VAL(P_n) found by the algorithm are equal to the values VAL(FALSE), ..., VAL(P_n) found by Dowling and Gallier's bottom-up algorithm [1] on input G_S. In fact the analysis of MODIFIED TOP-DOWN shows that it has two alternating phases:

(1) The algorithm visits G_S in a top-down fashion from FALSE to TRUE by discovering the presence of cycles thanks to SUSP fields. In this way it determines a subgraph, G_N, defined by the nodes in N and by all the visited edges.

(2) For each P in N the algorithm evaluates VAL(P) by visiting G_N in a bottom-up fashion, beginning from the nodes in N'. In fact the procedure EVALUE propagates the value *true* from N' to FALSE in the backtracking by disregarding all the edges ingoing in nodes with SUSP equal to *true* (i.e. by disregarding all the cycles), and propagates the value *true* in the cycles thanks to bottom-up visits of subgraphs corresponding to these nodes.

Therefore from the correctness of Dowling and Gallier's bottom-up algorithm it follows that MODIFIED TOP-DOWN correctly finds VAL(P) for each visited node P.

It follows that VAL(FALSE) is true if and only if S is unsatisfiable. □

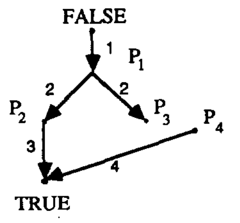

FIGURE 6.

It is immediate to observe that the algorithm MODIFIED TOP-DOWN is linear. In fact each edge of G_S is visited two times at most, once top-down and once bottom-up.

We also observe that if S is satisfiable, the algorithm may find a partial truth assignment. Let us consider for instance $S = \{\neg P_1, P_1 \vee \neg P_2 \vee \neg P_3, P_2, P_4\}$. The graph G_S corresponding to S is shown in Figure 6.

The algorithm correctly decides that S is satisfiable, but EVALUE(P_4) is not called and therefore VAL(P_4) remains *false*. In this case it is sufficient to call EVALUE(P) for each node P never visited [i.e. with COMPUTED(P) = *false*] to find the least truth assignment satisfying S in the boolean algebra $\{false, true\}^k$, where k is the number of distinct propositional letters in S [1].

We are grateful to G. Gallo for his helpful comments.

REFERENCES

1. Dowling, W. F. and Gallier, J. H., Linear-Time Algorithms for Testing the Satisfiability of Propositional Horn Formulae, *J. Logic Programming* (1984).
2. Gallo, G. and Urbani, G., Algorithms for Testing the Satisfiability of Propositional Formulae, Dipartimento di Informatica, Univ. di Pisa, 1987; *J. Logic Programming*, to appear.