

# A tautology checker for propositional formulae

M. Richards  
Computer Laboratory  
University of Cambridge  
`mr@cl.cam.ac.uk`  
`www.cl.cam.ac.uk/users/mr/`

May 15, 2006

## Abstract

This paper presents an algorithm to determine whether a Boolean formula always evaluates to false (or true) for all possible settings of its variables. The algorithm is based Stålmarck's proof procedure. It uses a set of Boolean relations over up to eight Boolean variables to represent the given formula, and a Boolean matrix to hold derived dyadic implications such as  $x \rightarrow y$  and  $x \rightarrow \bar{y}$ . A vector is used to hold equality relations such as  $x = 0$ ,  $x = 1$ ,  $x = y$  and  $x = \bar{y}$ . It applies rules to simplify the set of relations and deduce the dyadic implications, which are analysed to generate equality relations that are then applied to the relation set. When no more progress can be made, a variant of Stålmarck's dilemma rule is applied. Compared with Stålmarck, this algorithm represents the given formula using fewer terms and fewer variables, and has available more simplification and inference rules. The use of dyadic implications allows it to make more progress before having to resort to the potential expensive dilemma rule. In addition, it can be implemented efficiently using bit pattern techniques.

**Keywords.** Tautology checking, Boolean satisfiability, circuit verification.

## 1 Introduction

Both tautology checking and satisfiability testing are well known to have many useful applications and there are several published algorithms that attempt to solve these problems. This paper outlines a new algorithm to solve these problems based on Stålmarck's proof procedure[5, 2, 3]

Stålmarck's algorithm represents the given formula by a set of terms of the form  $x = (yoz)$  where  $o$  is a Boolean function and  $x$ ,  $y$  and  $z$  are positive or negative Boolean variables or the constants 0 or 1. In one version of the algorithm,  $o$  is limited to just the implies operator. Several rules are available to simplify the terms and to deduce equality relations of the form:  $x = 0$ ,  $x = 1$ ,  $x = y$  or  $x = \bar{y}$ . For example, the term  $1 = (1 \vee z)$  can be eliminated

since it is always satisfied, and, from  $x = (0 \rightarrow z)$ , we can deduce that  $x = 1$ . Deductions can also be made by taking terms in pairs. For example, from  $a = (x \wedge y)$  and  $b = (x \wedge y)$ , we can deduce  $a = b$ , and if we know the  $a = b$ , the variable  $a$  can be replaced by  $b$  in every term, possibly allowing further simplifications and deductions to be made. If an unsatisfiable term is ever found, the given formula is unsatisfiable. If the set of terms becomes empty, the given formula is satisfiable. If neither of these occur and no further simplification is possible, the algorithm applies the Dilemma rule. This involves selecting  $n$  variables and applying the algorithm to the  $2^n$  subproblems formed by all the possible settings of these  $n$  variables. If all are unsatisfiable the given formula is unsatisfiable. If any is satisfiable, the given formula is satisfiable, and if all those subproblems that are not unsatisfiable imply, say,  $x = y$ , then this is true in the parent problem. The number  $n$  is called the recursion depth and the algorithm attempts to solve the problem with successively increasing values of  $n$ . In practise, many large problems can be solved with quite small values of  $n$ .

The new algorithm extends Stålmarck's algorithm in several ways. Firstly, the set of terms is replaced by a set of Boolean relations. Notice that a Stålmarck term such as  $x = (y \wedge z)$  can be regarded as a relation over three variables specifying which of the eight settings of  $xyz$  are allowed. In this case they are: 000, 001, 010 and 111. This set can be represent by a bit pattern of length eight. It is natural to extend Stålmarck terms to all 256 possible relations of three variable rather than just the 16 possible Boolean functions, and a further extension is to increase the number of variables allowed in the relations. There are advantages and disadvantages in increasing this number, but a reasonable compromise seems to be relations over eight variables. This can be represented using a relation bit pattern of length 256 and eight integers to identify the variables. We will see how to use these relations later. A second extension to Stålmarck's algorithm is the use of a Boolean matrix to hold implications of the form:  $x \rightarrow y$ ,  $x \rightarrow \bar{y}$ ,  $\bar{x} \rightarrow y$  or  $\bar{x} \rightarrow \bar{y}$ . From transitive closure of this matrix, it is easy to deduce equality relations of the form:  $x = 0$ ,  $x = 1$ ,  $x = y$  or  $x = \bar{y}$ , and these can be used to eliminate variables from the original relation set. Using these larger terms reduces their number and the number of variables required. It also increases the number of simplifications and inference rules available. Furthermore, most manipulations of these terms can be done efficiently using bit pattern techniques.

The dilemma rule of the new algorithm selects a pivot relation that has many variables and few ones in its relation bit pattern. The subproblems considered correspond to each one occurring in the bit pattern. If the pivot relation has eight variables, we typically have less than 40 of the 256 subproblems to consider. For problems requiring significant recursion depth, this is a major advantage. The choice of pivot must be made with care since it is a compromise between the number of ones in its bit pattern and the cumulative frequency of use of its variables in the other relations.

As with many published algorithms, success depends on the details of the implementation, so some of these details are outlined in the following sections.

## 1.1 Notation

Let  $f(v_1, \dots, v_n)$  be a Boolean function of  $n$  Boolean variables. If there are no possible settings of the variables  $v_1, \dots, v_n$  for which  $f(v_1, \dots, v_n) = 0$ , then  $f$  is a tautology, and if there are no possible settings of the variables for which  $f(v_1, \dots, v_n) = 1$ , then  $f$  is not satisfiable. The algorithm described in this paper can determine whether either of these properties hold. Satisfiability and tautology checking are well known to be NP and co-NP-Complete[1], respectively, and so any algorithm to solve these problems will sometimes take exponential time. However, many Boolean formulae arising from real applications can be solved much more quickly.

We will assume, without loss of generality, that the function  $f$  is given as a directed acyclic graph in which each internal node contains a monadic or dyadic Boolean functions and each leaf edge is labelled by a variable from the argument list of  $f$ . The internal edges are then given other distinct labels and the leading edge, denoting the result of the function, is given the special label *res*. The graph is then partitioned in such a way that each partition has no more than eight edges entering or leaving it. These partitions can be represented by Boolean relations constraining the possible values on their external edges. The entire function can thus be represented as a set of relations over no more than eight variables.

It is common to specify benchmark problems using Conjunctive Normal Form (CNF). These are easily converted into relations over no more than eight variables by the addition of a few extra variables. Unfortunately the resulting relations have a high density of ones in their bit patterns which is not a good starting point for the new algorithm.

To test whether  $f$  is a tautology, we set the value of *res* to 0 and test whether the resulting relations lead to an inconsistency. Alternatively, if we set *res* to 1 and find an inconsistency the relations are not simultaneously satisfiable. Either problem thus reduces to searching for an inconsistency in a set of relations.

## 2 The internal representation

A relation  $R(a, b, c, d, e, f, g, h)$  over eight Boolean variables  $a \dots h$  is represented internally by a bit pattern  $w$  of length 256 and a vector of integers identifying the variables. With  $i = a + 2b + 4c + 8d + 16e + 32f + 64g + 128h$ , bit  $i$  of  $w$  is 1 if and only if the values of  $a, \dots, h$  satisfy the relation. The bits are numbered from the right with the rightmost bit numbered zero. Of course in a practical implementation, relations nodes have other fields to improve efficiency.

Relations over fewer than eight variables can be extended to a relation over eight variables by adding dummy zeros to the argument list. If the last few variables of a relation are zero they can be omitted from the written form. So, for instance,  $R(a, b, c, d)$  denotes a relation  $R(a, b, c, d, 0, 0, 0, 0)$ . Sometimes the name of a relation includes its bit pattern specified in binary or hexadecimal. For example,  $R10010101(x, y, z)$  denote the relation in which  $xyz$  must be one of 111, 001, 010 or 000 as shown in Fig 1.

A relation is written (or read) as a bit pattern composed of up to eight 32-bit

x	1 0 1 0 1 0 1 0
y	1 1 0 0 1 1 0 0
z	1 1 1 1 0 0 0 0
bit pattern	1 0 0 1 0 1 0 1
bit number	7 6 5 4 3 2 1 0

Figure 1:  $R10001101(x, y, z)$  represents  $(xyz \vee \bar{x}\bar{y}z \vee \bar{x}y\bar{z} \vee \bar{x}\bar{y}\bar{z})$

hexadecimal numbers followed by the variables. If fewer than eight numbers are given the bit pattern is padded to the left with zeros. Each variable appears as  $v$  followed by its non negative variable number. By convention, the variable  $v_0$  always has value 0 representing false. Comments start with a hash (#) and run to the end of the line. Using this notation, the Boolean formulae  $v_1 = (v_2 \rightarrow v_3)$  and  $(v_1 \vee v_2 \vee v_3) \rightarrow (v_4 \wedge v_5)$  can be written as:

```
A6 v1 v2 v3          # v1 = (v2->v3)
FF010101 v1 v2 v3 v4 v5 # (v1|v2|v3)->(v4&v5)
```

The following is an example of a relation over seven variables.

```
AB01007F 0 0 C81F0073 v1 v2 v3 v4 v5 v6 v7
```

It specifies the possible settings the variables  $v_1, \dots, v_7$  may take. In this example, bits 0 to 31 of the relation bit pattern are given by **C81F0073**, the next 64 bits are all zero and bits 96 to 127 are given by **AB01007F**. Notice that, if  $(v_6, v_7)$  is  $(0, 0)$ , the above relation reduces to  $RC81F0073(v_1, v_2, v_3, v_4, v_5)$ , and if  $(v_6, v_7)$  is  $(1, 1)$  then  $RAB01007F(v_1, v_2, v_3, v_4, v_5)$  holds, and that no other settings of  $(v_6, v_7)$  are not permitted. We can therefore deduce that  $v_6 = v_7$ . Careful inspection will reveal that this relation also implies  $v_4 \rightarrow v_5$ . Such implications can be detected efficiently by testing that particular subsets of the relation bit pattern contain only zeroes.

### 3 Simplification

Simplifications are applied to reduce the number of relations in the set, the total numbers of variables used, or the number of ones occurring in relation bit patterns. The simplification rules are applied either to a single relation or pairs of relations. Although the examples given typically involve relations over three variables, they are easily generalised to relations over eight variables.

#### 3.1 Simplification of a single relation

These simplifications apply to a single relation, but occasionally they make use of extra information such as  $x \rightarrow y$  or  $x = 1$ .

**FALSE** If a relation has bit pattern entirely composed of zeroes, it can never be satisfied and so simplification is terminated with an indication that the current problem is unsatisfiable.

**TRUE** A relation can be eliminated if it is satisfied for all settings of its variables. For example,  $R11111111(x, y, z)$  is always satisfied and so can be eliminated.

**DUP** If two of the arguments of a relation are equal then one of them can be eliminated. For example, the bit pattern of  $R10011101(x, y, x)$  states that  $xyx$  must be one of 111, 001, 110, 010 or 000, but we know that the first and third arguments are the same so the choice is reduced to 111, 010 or 000 which corresponds to a relation  $R1011(x, y)$  restricting  $xy$  to one of 11, 01 or 00. This rule is closely related to EQ, defined below, in which two different variables of a relation are known to have the same value.

**SET** If an argument variable has a known value, it can be removed from the relation with a suitable change of bit pattern. For example, consider the relation  $R10010101(x, y, z)$  which restricts to  $xyz$  to one of 111, 001, 010 or 000. If  $z$  is known to be 0, the possible settings of  $xy$  is restricted to 01 and 00, and so the simplified relation is  $R0101(x, y)$ . Similarly, if  $z$  is known to be 1, the relation reduces to  $R1001(x, y)$ . Note that 1001 and 0101 are the two halves of 10010101.

**EQ** If the values of two arguments of a relation are known to be equal, one of them can be eliminated. For example, the relation  $R10010101(x, y, z)$  which restricts to  $xyz$  to one of 111, 001, 010 or 000 reduces to  $R1011(x, y)$  if the value of  $x$  is known to be equal to  $z$ . This restricts  $xy$  to be one of 11, 01 or 00.

**NE** If the values of two arguments of a relation are known to be unequal, one of them can be eliminated. For example, the relation  $R10010101(x, y, z)$  which restricts to  $xyz$  to one of 111, 001, 010 or 000 reduces to  $R0001(x, y)$ , if it is known that  $x = \bar{z}$ . This specifies that  $xy$  can only be 00.

**IMP** If the values of two arguments of a relation,  $x$  and  $y$  say, are known to satisfy  $x \rightarrow y$ , the relation can be simplified. For example, consider the relation  $R10010101(x, y, z)$  which restricts to  $xyz$  to one of 111, 001, 010 or 000. If it is known that  $x \rightarrow z$  then the triplet 001 is disallowed, reducing the possible settings of  $xyz$  to one of 111, 010 or 000. The simplified relation is thus  $R10000101(x, y, z)$ . The other three implications,  $x \rightarrow \bar{y}$ ,  $\bar{x} \rightarrow y$ , or  $\bar{x} \rightarrow \bar{y}$  give similar simplifications.

**SINGLE** If a variable is used in only one relation, its value is unrestricted by the other relations and so can be removed. For example, if  $z$  is only used in  $R10010101(x, y, z)$  which restricts  $xyz$  to being one of 111, 001, 010 or 000, the relation can be simplified to  $R1011(x, y)$  which restricts  $xy$  to one of 11, 10 or 00.

**PERM** This rule allows the variables of a relation to be reordered. This involves a permutation of the relation bit pattern. For example, the relation  $R10011101(x, z, y)$  specifies that  $xzy$  must be one of 111, 001, 110, 010

or 000. This is equivalent to saying that  $xyz$  must be one of 111, 010, 101, 001 or 000 which is represented by  $R10110101(x, y, z)$ .

**SPLIT** If a relation  $R$  can be factorised into two relations  $S$  and  $T$  that have no variables in common, then  $R$  should be replaced by these two relations. For instance,  $RA700A7A7(a, b, d, c, e)$  can be split into  $R10100111(a, b, c)$  and  $R1011(d, e)$ . SPLIT is expensive and rarely succeeds, and so is only applied when all other simplifications have been applied and then only to relations over six or more more variables, since monadic and dyadic relations will have already been extracted by the inference rules described below. Splitting is useful since the resulting factors may sometimes be combined with other relations using the COMBINE simplification described below.

### 3.2 Pairwise simplifications

There are two simplifications involve pairs of relations, as follows.

**COMBINE** If the total number of variables used in two relations is no more than eight and they have at least one variable in common, they can be combined to form a single relation. For example,  $R10010101(a, y, z)$  which restricts to  $ayz$  to one of 111, 100, 010 or 000 can be combined with  $R10111011(b, y, z)$  corresponding to  $byz$  being one of 111, 101, 100, 011, 001 and 000 resulting in  $R00001100100010001(a, b, x, y)$  corresponding to  $abxyz$  being one of 1101, 1010, 1000, 0100 and 0000. In general its is best to combine relations that have many variables in common. COMBINE is particularly cheap to implement when two relations use the same variables in the same order. For instance  $R4F013A04(a, b, c, d, e)$  and  $RC57200FF(a, b, c, d, e)$  can be replaced by  $R45000004(a, b, c, d, e)$ , since  $45000004 = (4F013A04 \& C57200FF)$ .

**RESTRICT** If COMBINE cannot be used because the resulting relation would have too many variables, it may still be possible for one relation to restrict another. If two relations  $S$  and  $T$  have some variables in common and if  $S$  imposes a restriction on the possible settings of these shared variables then this restriction can be applied to  $T$  replacing it by a simpler relation. For example, if  $S$  is  $R10000111(a, y, z)$  which restricts to  $ayz$  to one of 111, 010, 100 or 000 which thus restricts  $yz$  to be one of 11, 10 or 00, and if  $T$  is  $R10111000(b, y, z)$  corresponding to  $byz$  being one of 111, 101, 001 and 110, the restriction implied by  $S$  limits these possibilities to just 111 and 110 corresponding to the new version of  $T$  being  $R10001000(b, y, z)$ . Note that,  $T$  can also restrict  $S$ , replacing it by  $R10000100(a, y, z)$ . RESTRICT is only applied to pairs of relations having three or more variables in common, since restrictions between pairs of variables will have already been extracted and applied by the inference rules described below. RESTRICT can be implemented in a way that allows the following implications to be detected:  $x \rightarrow y$ ,  $x \rightarrow \bar{y}$ ,  $\bar{x} \rightarrow y$ , or  $\bar{x} \rightarrow \bar{y}$  where  $x$  is only in  $S$  and  $y$  only in  $T$ .

## 4 Inferences

After the relations have been simplified they are inspected to find all the dyadic relations of the form  $x \rightarrow y$ ,  $x \rightarrow \bar{y}$ ,  $\bar{x} \rightarrow y$ , or  $\bar{x} \rightarrow \bar{y}$  that they imply. As we have seen these are easily discovered by applying a mask to the relation bit pattern and testing for zero. For example, since  $(95C1008D \ \& \ 0000FF00)$  is zero, we can deduce that  $d \rightarrow e$  is implied by  $R95C1008D(a, b, c, d, e)$ . Notice that this relation also implies  $a \rightarrow b$  since  $(95C1008D \ \& \ 22222222)$  is zero. These dyadic relations are stored in a matrix as described in the next section. Although it is possible to deduce particularly useful relations such as  $v_i = \bar{v}_j$  or  $v_i = 0$  using bit pattern techniques on the relation bit pattern, this is not done since such facts are discovered more efficiently from the matrix.

For relations over eight variables there are 28 variable pairs to consider, and for each pair  $(x, y)$ , say, there are four possible implications:  $x \rightarrow y$ ,  $x \rightarrow \bar{y}$ ,  $\bar{x} \rightarrow y$  or  $\bar{x} \rightarrow \bar{y}$ . Since  $(x \rightarrow y) = (\bar{x} \vee y) = (\bar{y} \rightarrow \bar{x})$ , we need only consider pairs in which the variable number of  $x$  is greater than that of  $y$ . There are thus 112 possible dyadic implications, and these can be packed as bits in four 32-bit words. Each 1 occurring in the relation bit pattern causes 28 of the 112 implications bits to be set to zero. Every other implication bit should remain set to 1. The computation proceeds by considering the relation bit pattern one byte at a time and masking out the implication bits disallowed by the ones in that byte. This can be done efficiently using precomputed tables of 256 112-bit masks for each of the 32 byte positions in the relation bit pattern. Since relation bit patterns often contains many zero words, only bytes occurring in non zero words are processed. Relation nodes hold previous implication bit patterns so newly discovered implications are easy to detect.

## 5 The Boolean matrix

The dyadic implications between pairs of variables are stored in a  $2n \times 2n$  Boolean matrix  $M$  which is partitioned into four  $n \times n$  Boolean matrices  $A$ ,  $B$ ,  $C$  and  $D$  as follows:

$$M = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$$

$A_{ij}$  is set to 1 if the implication  $v_i \rightarrow v_j$  is known to hold. Similarly,  $B_{ij}$ ,  $C_{ij}$  and  $D_{ij}$  are respectively set to 1 if  $v_i \rightarrow \bar{v}_j$ ,  $\bar{v}_i \rightarrow v_j$  or  $\bar{v}_i \rightarrow \bar{v}_j$  are known. Note that  $D$  is the transpose of  $A$  since  $(v_i \rightarrow v_j) = (\bar{v}_j \rightarrow \bar{v}_i)$ , and for similar reasons  $B$  and  $C$  are symmetric.

When all the newly discovered implications have been stored to  $M$ , a variant of Warshall's algorithm is applied to form its transitive closure. Warshall's algorithm was originally presented in [6] but is also described in many text books such as [1]. The variant used here takes advantage of the known symmetry properties of  $M$  and knowledge of which implications have been added since the previous transitive closure.

From the resulting matrix, we can easily determine relations of the form  $v_i = v_j$ ,  $v_i = \bar{v}_j$ ,  $v_i = 0$  and  $v_i = 1$  since

$$\begin{aligned}
(A \wedge D)_{ij} = 1 &\implies (v_i \rightarrow v_j) \wedge (\overline{v_i} \rightarrow \overline{v_j}) \implies v_i = v_j \\
(B \wedge C)_{ij} = 1 &\implies (v_i \rightarrow \overline{v_j}) \wedge (\overline{v_i} \rightarrow v_j) \implies v_i = \overline{v_j} \\
(A \wedge B)_{ij} = 1 &\implies (v_i \rightarrow v_j) \wedge (v_i \rightarrow \overline{v_j}) \implies v_i = 0 \\
(C \wedge D)_{ij} = 1 &\implies (\overline{v_i} \rightarrow v_j) \wedge (\overline{v_i} \rightarrow \overline{v_j}) \implies v_i = 1.
\end{aligned}$$

Each of these equality relations is particularly useful since it allows the elimination of the variable  $v_i$  from every relation in the current set. If it is ever discovered that  $v_i = \overline{v_i}$  then an inconsistency has been found, the current set of relations is unsatisfiable. Newly discovered relations of the form  $v_i \rightarrow v_j$ ,  $v_i \rightarrow \overline{v_j}$ ,  $\overline{v_i} \rightarrow v_j$  and  $\overline{v_i} \rightarrow \overline{v_j}$  can also be used to simplify relations in the current set using IMP defined above.

After all the newly discovered dyadic relations have been applied to the current set of relations, the algorithm returns to the simplification phase, possibly finding more previously undetected implications. The process continues until no more progress can be made. At this stage either an inconsistency will be found implying that the current set of relations is unsatisfiable, or the set of relations is empty, implying that they can be satisfied. If neither of these cases arise, SPLIT is applied to every relation over six or more variables and any resulting factors combined with other relations using COMBINE. Although this simplification is useful, it will not yield new implications and so the algorithm must now invoke the dilemma rule.

## 6 The dilemma rule

The algorithm applies the dilemma rule when the set of relations are (a) fully simplified, (b) contain no obvious inconsistencies and (c) from which no new implications can be deduced.

It involves choosing a pivot relation from the current set, probably one containing many frequently used variables and reasonably few ones in its relation bit pattern. The ones in the bit pattern specifies the possible setting of its variables. Typically there will be, on average, no more than about 40 possible settings to consider. These are analysed in turn. If they all lead to inconsistencies then the original set of relations is unsatisfiable. If any case is satisfiable, the original set is satisfiable. If all the non-inconsistent cases imply that, say,  $x \rightarrow y$  holds then this relation can be applied to the original set. Similarly, for the other three implications. Once all these newly discovered implications have been applied to the original set of relations, the algorithm returns again to the simplification phase.

The depth of nesting of the sub-problems created by the dilemma rule is called the recursion depth. The maximum recursion depth allowed is successively increased until the problem is solved. Luckily most problems in practise can be solved with small depth limits.

## 7 The overall algorithm

The algorithm is encapsulated in the function:

*explore*(relations  $R$ , matrix  $M$ , int  $depth$ , int  $maxdepth$ )



which performs the following computation.

- (1) Apply the simplification and inference rules to  $R$ , inserting any detected implications of the form  $x \rightarrow y$ ,  $x \rightarrow \bar{y}$ ,  $\bar{x} \rightarrow y$  or  $\bar{x} \rightarrow \bar{y}$  into  $M$ . If an inconsistency is found, return from *explore* with an indication that the relations are not satisfiable. If the set of relations becomes empty, return from *explore* with an indication that the relations are satisfiable.
- (2) Form the transitive closure of  $M$  and apply any newly discovered relations of the form  $x = 0$ ,  $x = 1$ ,  $x = y$  or  $x = \bar{y}$  to  $R$ .
- (3) Repeat steps (1) and (2) until no more progress can be made.
- (4) If  $depth < maxdepth$ , return from *explore* with an indication that the current problem cannot be solved with this setting of *maxdepth*.
- (5) Apply SPLIT to every relation having six or more variables.
- (6) Choose a pivot relation  $P$  and make a new Boolean matrix  $I$  entirely filled with ones.
- (7) For each new set of relations  $R'$  composed of  $R$  modified by each possible setting of the variables permitted by  $P$ , copy  $M$  into a new Boolean matrix  $M'$  and call: *explore*( $P'$ ,  $M'$ ,  $depth+1$ , *maxdepth*). If all these sub-problems are unsatisfiable, return from *explore* with an indication that the relations  $R$  are not satisfiable. If any sub-problem is satisfiable return from *explore* with an indication that the relations  $R$  are satisfiable. Otherwise form the intersection in  $I$  of all the matrices returned by the calls of *explore* that did not indicate unsatisfaction.
- (8) If  $I$  contains implications that are not in  $M$ , apply them to  $R$  and continue processing from (1).
- (9) Otherwise, return from *explore* with an indication that the problem cannot be solved with this setting of *maxdepth*.

A useful optimisation is to rename the variables to form a compact set if they become too sparse, since this improves the efficiency of the matrix operations. If this compaction is done, it is best performed in step (7).

## 8 Comments

The implementation of this algorithm is not complete so no performance results are yet available but it is expected that its complexity will be similar to that of Ståmarck's algorithm, although it is hoped that it will perform better particularly on harder problems. The current state of a BCPL implementation of the algorithm can be found in directory `BCPL/bcplprog/chk8/` in the freely available BCPL distribution available via [4]. A implementation in C will be made available in due course.

If using relations over eight variables turns out to be of great benefit, relation over a larger number, 16 say, may be even better, but then the bit pattern representation of relations would no longer be economical. It may turn out that using Ordered Binary Decision Diagrams (OBDDs) would work well. Note that many of the well known problems with OBDDs are avoided here since they are guaranteed to be small since they only have a small number of variables. Other

ways to represent relations over a small number of variables are also worth considering.

## References

- [1] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 1990.
- [2] G. Stålmarck and M. Sjöflund. Modelling and verifying systems and software in propositional logic. In B.K. Daniels, editor, *Safety of Computer Control Systems, 1990(SAFECOMP'90)*, pages 31–36, 1990.
- [3] J. Harrison. Stålmarck's Algorithm as a HOL derived rule. In J. von Wright, J. Grundy and J. Harrison, editor, *Proceedings of TPHOLs'96*, Lecture Notes in Computer Science, pages 221–234. Springer-Verlag, 1996.
- [4] M. Richards. *The BCPL Cintcode Distribution*. [www.cl.cam.ac.uk/users/mr/BCPL/bcpl.{tgz,zip}](http://www.cl.cam.ac.uk/users/mr/BCPL/bcpl.{tgz,zip}).
- [5] Mary Sheeran and Gunnar Stålmarck. A tutorial on Stålmarck's proof procedure for propositional logic. In G. Gopalakrishnan and P. Windley, editors, *Proceedings 2nd Intl. Conf. on Formal Methods in Computer-Aided Design, FMCAD'98, Palo Alto, CA, USA, 4–6 Nov 1998*, volume 1522, pages 82–99. Springer-Verlag, Berlin, 1998.
- [6] S. Warshall. A theorem on boolean matrices. *J. ACM*, 9:11–12, 1962.