******* Very Rough Draft ********
Experiments with a Tautology Checking Algorithm

by

Martin Richards[1]

mr@uk.ac.cam.cl
http://www.cl.cam.ac.uk/users/mr/

Computer Laboratory
University of Cambridge

September 15, 2005

Abstract

This paper presents various versions of a tautology checking algorithm that is related both to Stålmarck's algorithm and the mechanism described by Kunz and Stoffel in *Reasoning in Boolean Networks*[KS97]. The algorithm uses as its central data structure the conjunction of a set of terms, with each term being a relation over a small number of variables. In Stålmarck's algorithm the relation is limited to Boolean operators with up to two operand variables and one result variable. This representation has been generalised to allow arbitrary relations on three boolean variables, and then further extended to allow relations over as many as eight boolean variables.

Another extension is to increase the richness of the variable mapping information obtained from the inference rule applications. In Stålmarck's algorithm the mapping information is a set of equalities/inequalities between pairs of variables. This has been extended to arbitrary relations over boolean pairs.

The paper includes test results showing the effect these various extensions have depth of recursion needed and the number of terms required at each level.

Keywords

Tautology checking, Boolean satisfiability, Boolean networks, digital circuit verification, MCPL

[1]Currently visiting: School of Computer Science, Carnegie Mellon University, Pittsburgh

Acknowledgements

I gratefully acknowledge the encouragements and the many helpful discussions I have had with Ed Clarke and Randy Bryant in the course of this work.

1 Introduction

Tautology checking is useful — Applications, previous work — Mention Stalmarck's Algorithm and Recursive learning.

2 Stalmarck's Algorithm

In essence, Stalmarck's algorithm works as follows. Take a given propositional calculus expression, E say, consisting of propositional variables $(v_1, v_2, \ldots)$ and Boolean operators, and convert it into a set of terms of the form $op(t_i, a, b)$, one for each subexpression of E. The first argument t_i is a newly assigned variable holding the result of the subexpession, and the other two arguments are either subterm variables or propositional variables. From now on, we will use $v_1, v_2, \ldots, v_n$ to identify both subterm and propositional variables. We will let *op* range over all of the 16 possible dyadic Boolean operators. Applying this transformation to the following propositional expression:

$$((a \equiv b) \rightarrow c) \wedge (\neg(b \equiv a) \vee c)$$

yields the following set of terms:

$$\begin{array}{l} eqv(t_1, a, b) \\ imp(t_2, t_1, c) \\ eqv(t_3, b, a) \\ noty(t_4, t_3, 0) \\ or(t_5, t_4, c) \\ and(t_6, t_2, t_5) \end{array}$$

where *eqv*, *imp*, *noty*, *or* and *and* denote relations between the arguments and results of the corresponding boolean operators. For instance,

$$\begin{array}{l} eqv(x, y, z) = (x \equiv (y \equiv z)) \\ imp(x, y, z) = (x \equiv (y \rightarrow z)) \\ noty(x, y, z) = (x \equiv (\neg y)) \\ or(x, y, z) = (x \equiv (y \vee z)) \\ and(x, y, z) = (x \equiv (y \wedge z)) \end{array}$$

To test whether the original expression is a tautology, the root variable (t_6 in this case) is set to 0 denoting false, and then the set of terms analysed to see if this leads to an inconsistency. If it does then the original expression is a tautology, otherwise it is possible to assign values to all the variables in such a way that every term is satisfied. During the analysis, information about possible variable assignments is built up. This variable mapping information is held as a set of items of the form: $v_i = 0, v_i = 1, v_i = v_j$ or $v_i = \bar{v_j}$. Whenever an item is added to this set, it is immediately used to simplify the set of terms. However, a new item may be inconsistent with items already in the mapping. For instance, $v_i = v_j$ and $v_i = \bar{v_j}$ cannot simultaneously be satisfied. If such an inconsistency arises then the set of terms cannot be satisfied.

There are several obvious inference rules that can be applied to the terms to simplify them and to generate new items in the variable mapping set. In this example, the term $noty(t_4, t_3, 0)$ generated the mapping item $t_4 = \bar{t_3}$, and this simplifies the terms to:

$eqv(t_1, a, b)$
$imp(t_2, t_1, c)$
$eqv(t_3, b, a)$
$imp(t_5, t_3, c)$
$and(0, t_2, t_5)$

The terms $eqv(t_1, a, b)$ and $eqv(t_3, b, a)$ can be combined to yield the mapping $t_3 = t_1$, resulting in the following set of terms:

$eqv(t_1, a, b)$
$imp(t_2, t_1, c)$
$imp(t_5, t_1, c)$
$and(0, t_2, t_5)$

The terms $imp(t_2, t_1, c)$ and $imp(t_5, t_1, c)$ can be combined to yield the mapping $t_5 = t_2$, resulting in the following set of terms:

$eqv(t_1, a, b)$
$imp(t_2, t_1, c)$
$and(0, t_2, t_2)$

The term $and(0, t_2, t_2)$ gives $t_2 = 0$, resulting in:

$eqv(t_1, a, b)$
$imp(0, t_1, c)$

The term $imp(0, t_1, c)$ gives $t_1 = 1$ and $c = 0$, resulting in:

$eqv(1, a, b)$

and finally, this give $a = b$ resulting in the empty set of terms. The original expression was thus not a tautology.

In general we cannot expect to be so lucky. For instance, if we apply the same transformation to the expression:
$((a \vee b) \wedge (\neg a \vee b) \wedge (a \vee \neg b)) \rightarrow \neg(\neg a \vee \neg b)$
we obtain the following terminal set of terms:

$and(0, a, b)$
$lt(0, a, b)$
$gt(0, a, b)$
$nor(0, a, b)$

Here, $lt(x, y, z) = (x \equiv (y = 0 \wedge z = 1))$, and $gt(x, y, z) = (x \equiv (y = 1 \wedge z = 0))$. The first term restricts the possible settings of ab to 00, 01 and 10. The second to 00,10 and 11, the third to 00,01 and 11, and the fourth to 01,10 and 11, so clearly no setting of ab can satisfy all four terms. Unfortunately, with this kind of term set and form of variable mapping, no direct inferences are applicable to these terms. The solution to in Stalmarck's algorithm is to apply the dilemma rule.

2.1 The dilemma rule

Select a variable v_i, say, that has not already been assigned a value and try setting it to 0 and 1 in turn, and apply the direct inferences in each case until convergence. This will give two new variable mappings M_0 and M_1, say. If an inconsistency was discovered while forming either of these, a special result ω is returned. A new mapping M is now formed by taking the intersection of M_0 and M_1, consisting of mapping items that occur in both mappings. However, if both M_0 and M_1 are ω then no consistent mapping is possible, and, if either M_0 or M_1 is ω, the other is returned. As an example of the general case, suppose $M_0 = \{v_1 = 0, v_2 = 1, v_3 = \bar{v_4}, v_5 = v_6\}$ and $M_1 = \{v_1 = 0, v_2 = 0, v_3 = v_5, v_5 = v_6\}$ then the intersection $M = \{v_1 = 0, v_5 = v_6\}$.

This process is repeated on every remaining variable in turn. Often a solution is found during this process either by encountering a variable for which both $M_1 = \omega$ and $M_2 = \omega$, or by the set of remaining terms becoming empty, but in general this will not happen. If this happens, the dilemma rule will have to be applied to all pairs of unset variables. For each pair (v_i, v_j), this involves intersecting four possible mapping M_{00}, M_{01}, M_{10} and M_{11}, corresponding the four possible settings of (v_i, v_j). It may be found necessary to apply the dilemma rule to triplets or even larger sets of variables before a solution is found. The *recursion depth* of the dilemma rule is the number of variables simulataneously set during its iteration. In many practical applications a solution can be found before this depth reaches an unacceptable level. Results presented in section 11 give statistics of how the algorithm performs on benchmark tests up to a maximam depth of 5.

The direct derivations can be performed in polynomial time, but the dilemma rule has exponential time complexity, and so any scheme that potentially reduces the recursion depth is likely to be advantageous. One such scheme is to use general ternary relations over boolean variables, rather restricting them to the 16 dyadic boolean operators. This is developed in the next section.

3 Using ternary relations

There are 256 relations over three boolean variables. They can be represented using bit pattern of length eight shown in figure 1.

We give names to these relations of the form *Rabcdefgh*, so for example

and	=R10000111
lt	=R00101101
gt	=R01001011
nor	=R00011110

Returning to our previous example, the terms were:

$and(0, a, b)$
$lt(0, a, b)$
$gt(0, a, b)$
$nor(0, a, b)$

$$\begin{array}{ccccccccccl}
x & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & & \\
y & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & & \\
z & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & & \\
 & a & b & c & d & e & f & g & h & \text{The bit pattern} & \\
 & & & & & & & & \multicolumn{3}{l}{h = 1 \Leftrightarrow 000 \epsilon R} \\
 & & & & & & & \multicolumn{3}{l}{g = 1 \Leftrightarrow 001 \epsilon R} & \\
 & & & & & & \multicolumn{3}{l}{f = 1 \Leftrightarrow 010 \epsilon R} & & \\
 & & & & & \multicolumn{3}{l}{e = 1 \Leftrightarrow 011 \epsilon R} & & & \\
 & & & & \multicolumn{3}{l}{d = 1 \Leftrightarrow 100 \epsilon R} & & & & \\
 & & & \multicolumn{3}{l}{c = 1 \Leftrightarrow 101 \epsilon R} & & & & & \\
 & & \multicolumn{3}{l}{b = 1 \Leftrightarrow 110 \epsilon R} & & & & & & \\
 & \multicolumn{3}{l}{a = 1 \Leftrightarrow 111 \epsilon R} & & & & & & &
\end{array}$$

Figure 1: Bit pattern representation of a relation $R(x, y, z)$

We can now combine $and(0, a, b)$ and $lt(0, a, b)$ (taking the bit-wise *and* of their relations $R10000111$ and $R00101101$) giving:

$R00000101(0, a, b)$
$gt(0, a, b)$
$nor(0, a, b)$

The first two term can be combined to give:

$R00000001(0, a, b)$
$nor(0, a, b)$

and these two can be combined to give:

$R00000000(0, a, b)$

which cannot be satisfied. The original propositional expression is thus a tautology. We have thus solved this problem without having to resort to the dilemma rule.

Having as many as 256 relations may seem to be a disadvantage but there are compensations, in addition to the expected reduction in recursion depth. With 8-bit relations, terms can be canonicalised by the following steps.

1. Replace $r(x, y, z)$ by $r(0, y, z)$, if the relation does not depend on x. For example, replace $R10111011(x, y, z)$ by $R10111011(0, y, z)$, since the upper and lower 4 bits of r are equal.

2. Replace $r(1, y, z)$ by the equivalent $r'(0, y, z)$. For example, replace $R00110101(1, y, z)$ by $01010011(0, y, z)$, i.e. swap the upper and lowers 4 bits of r.

3. Replace $r(x, y, y)$ by the equivalent $r'(x, y, 0)$. For example, replace $R00110101(x, y, y)$ by $R00010001(x, y, 0)$, i.e. mask r with $R10011001$.

4. Replace $r(0, y, z)$ by the equivalent $r'(0, y, z)$. For example, replace $R00110101(0, y, z)$ by $R00000101(0, y, z)$, i.e. mask r with $R00001111$.

5. Do the above transformations for all permutations of (x, y, z).

6. Return $r(x, y, z)$ with the variables (x, y, z) in dictionary order.

The resulting term will have one of the following forms:

$$\begin{array}{l} Rabcdefgh(x,y,z) \\ R0000abcd(0,y,z) \\ R000000ab(0,0,z) \\ R0000000a(0,0,0) \end{array}$$

3.1 Patterns

The arguments variables of a term can have one of 37 possible patterns, if we take into account which are explicitly 0 or 1, and which variables are repeated. These patterns are a subset of the relations described above and can be represented in the same way. They are given mnemonic names such as $Px1z, Pxyy$ and $Pxyz$. It is easy to see that:

$$\begin{array}{ll} Px1z = R11001100 & \{111, 110, 011, 010\} \\ Pxyy = R10011001 & \{111, 100, 011, 000\} \\ Pxyz = R11111111 & \{111, 110, 101, 100, 011, 010, 001, 000\} \end{array}$$

3.2 Inferences

A general term $r(x, y, z)$ thus has bit patterns for its relation and the pattern for its arguments. From these it is possible to deduce the variables mappings implied by the relation that are implicitly the result of the pattern. This can be done efficiently by table lookup, and so provides a cheap mechanism finding inferences deducible from an individual term.

More inferences can be made by considering terms in pairs. If there are two terms $r(x, y, z)$ and $s(x, y, z)$ with identical arguments, they can be replaced by $t(x, y, z)$ where t is the bit pattern *and* of r and s. We have already seen an example of this inference rule above. If, however, the two terms have only two arguments in common, we can still make deductions. Suppose the terms are:

$$\begin{array}{ll} R01010011(a,x,y) & \{110, 100, 001, 000\} \\ R10101100(b,x,y) & \{111, 101, 011, 010\} \end{array}$$

The first disallows the pattern $xy = 11$ and the second disallows $xy = 00$, so, taken together we can deduce:

$$\begin{array}{ll} R01000010(a,x,y) & \{110, 001\} \\ R00100100(b,x,y) & \{101, 010\} \end{array}$$

From these we can deduce information about ab, namely $a = \bar{b}$, and taken separately the terms allow us to deduce new information about axy, (namely: $a = x = \bar{y}$), and bxy (namely: $b = \bar{x} = \bar{y}$). All these deductions can be made easily by means of bitwise logical operations on the relation bit patterns.

Another possible inference rule applies to a pair of terms of the form $r(x, 0, z)$ and $s(0, y, x)$, replacing them by $t(x, y, z)$. Again, the bit pattern representation of t is easily calculated from r and s.

4 Using larger relations

Having seen that it is easy to apply the inference rules using a bit pattern representation for the relations, it is natural to consider the effect of generalising the relations to work over 4, 5 or more Boolean variables. The length of the relation bit pattern is 2^n for relations over n variables, and the number of variable mapping items ($x = 0$, $x = 1$, $x = y$, $x = \bar{y}$) is $n + n + n(n-1)/2 + n(n-1)/2 = n(n+1)$. So when n is 4 there are 20 possible mapping items, and 16 bits are required to represent a relation. By the time n is 8 the relation takes eight 32 bit words to represent, which is typically equal to the space required to specify the arguments variables. A term can thus be represented in 16 words. This seems like a reasonable compromise. Note that terms over 6 variables would require 8 words for their representation.

It is worth noting that the relation need not be represented as a bit pattern. It would, for instance, be possible to use an Ordered Binary Decision Diagrams (OBDDs) for the purpose, opening the possibility of handling terms with up to 16, or possibly 32, variables with adequate efficiency, but in this paper only straightforward bit patterns representations are considered.

We will use the notation *Rn* to denote the set of terms containing general relations over n Boolean variables, and *BinOp* to denote the subset of $R3$ in which all terms are restricted to relations corresponding to the 16 dyadic Boolean functions. The variable mapping can be thought of a subset of $R2$ restricted to 12 of the 16 possible relations over two Booleans. The four that are omitted involve implication, possibly combined with negation. We denote this restricted set by *EqNe* since its terms contain information about the equality or inequality of variables compared to other variables or the constants 0 and 1. Its mapping terms are therefore of the form $x = 0, x = 1, x = y$ or $x = \bar{y}$. The four remaining mapping items in $R2$ are of the form $x \rightarrow y, x \rightarrow \bar{y}, \bar{x} \rightarrow y$ or $\bar{x} \rightarrow \bar{y}$, and we will denote this subset of $R2$ by *Imp*. Any set $R2$ can thus be partioned into two subsets *EqNe* and *Imp*.

It is possible that the terms of $R2$ are inconsistent, $R2 = \{x = 0, y = 1, \bar{x} \rightarrow \bar{y}\}$, for example. Such inconsistencies should be checked for whenever a new term is added to the set. Assuming no inconsistencies are present, the $EqNe$ set represents a mapping function which, when applied to all terms in the system, will eliminate all occurrences of the variables in its domain. For example, $\{x = \bar{y}$ and $z = 0$ in *EqNe* will eliminate x and z from all other terms, including those in Imp. Once this is done, these terms can be removed from $EqNe$, and we are left with only terms of the form $x \rightarrow y, x \rightarrow \bar{y}, \bar{x} \rightarrow y$ or $\bar{x} \rightarrow \bar{y}$, in which neither x nor y are 0 or 1. If these remaining terms contain a cycle then either the variables in the cycle are all equal or there is an inconsistency. For example:

$$\begin{array}{lcl} x \rightarrow y, \bar{y} \rightarrow z, z \rightarrow \bar{x} & \Longrightarrow & x = y = \bar{z} \\ x \rightarrow y, \bar{y} \rightarrow \bar{z}, z \rightarrow \bar{x} & \Longrightarrow & Inconsistent \end{array}$$

Such deductions should be made as soon as they are detectable. We are thus left with an $R2$ structure containing only terms from Imp that contain no cycles and we need an efficient algorithm to allow us to detect cycles when new items are added (to either Imp or $EqNe$). For our application the structure is likely to be sparse.

...

5 Extending the variable mapping

The mapping set *EqNe* can be represented using a vector, that maps variable identifiers to other, possibly negated, variable identifiers. If all the links in this vector refer to variables at lower subscript positions, the chains will be finite. Whenever a variable is looked up, its chain pointer can be replaced by the result of the look up, to improve the efficient subsequent look ups. This lazy optimization has the advantage that new mappings are cheap to add, while look ups have a once only chain following penalty.

It is natural to consider the effect of extending our mapping to include items from the whole of $R2$ rather than restricting it to the *EqNe* subset.

The efficient representation of Imp is less easy than $EqNe$, but it can still be done economically and it is well worth doing since it tends to reduce the recursion depth of the (exponentionally expensive) dilemma rule.

6 Algorithm Rn

.

Organisation – inference rules –flow of control decisions – implementation details.

The given propositional expression is first converted to terms in Rn. A typical term being $r(v_1, v_2, \ldots, v_n)$, and the root variable set to 0.

The rewrite and inference rules are:

6.1 PERM

$r(v_1, v_2, \ldots, v_n)$
$\Longrightarrow$ where $i, j, ...k$ is a permutation of $1, 2, ..., n$
$r'(v_i, v_j, \ldots, v_k)$

6.2 ELIMEQ

$r(v_1, v_1, v_3, \ldots, v_n)$
$\Longrightarrow$
$r'(v_1, 0, v_3, \ldots, v_n)$

By combining PERM and ELIMEQ, all repeated variables in a term can be removed.

6.3 ELIMVAR

$r(v_1, v_2, \ldots, v_n)$
$\Longrightarrow$ if v_1 is not used in any other term
$r'(0, v_2, \ldots, v_n)$

6.4 INDEP

$r(v_1, v_2, \ldots, v_n)$
$\Longrightarrow$ if the truth of this term does not depend on the value of v_1
$r'(0, v_2, \ldots, v_n)$

6.5 UNIT

$r(v_1, v_2, \ldots, v_n)$
$\Longrightarrow$
$r'(v_1, v_2, \ldots, v_n)$
$s(v_i, v_j)$

This rule should be applied whenever applicable to extract a dyadic relation from $r(v_1, v_2, \ldots, v_n)$. The extracted relation is $s(v_i, v_j)$, and $r'(v_1, v_2, \ldots, v_n)$ is $r(v_1, v_2, \ldots, v_n)$ with this constraint removed.

6.6 PAIR2

$s(a, b)$
$r(v_1, v_2, \ldots, v_n)$
$\Longrightarrow$
$s(a, b)$
$s'(v_i, v_j)$
$r'(v_1, v_2, \ldots, v_n)$

This rule combines a dyadic relation $s(a, b)$ with a term to produce a new dyadic relation $s'(v_i, v_j)$ and a replacement term $r'(v_1, v_2, \ldots, v_n)$ with this information removed. This rule is applied whenever applicable. The motivation is that it is always good to generate more dyadic terms. An example application of this rule is:
$\{x = 1\}\{x + y + z \leq 2, r + s + t \geq 1\} \Longrightarrow \{x = 1\}\{y + z \leq 1\}\{r + s + t \geq 1\}$

6.7 PAIRGEN

$r(v_1, v_2, \ldots, v_n)$
$s(w_1, w_2, \ldots, w_n)$
$\Longrightarrow$
$r'(v_1, v_2, \ldots, v_n)$
$s'(w_1, w_2, \ldots, w_n)$
$t(a, b)$

This rule should be applied whenever possible since it generates a dyadic term. The relations r' and s' are r and s with the information in $t(a, b)$ extracted.

6.8 PAIRSIMP

$r(v_1, v_2, \ldots, v_n)$
$s(w_1, w_2, \ldots, w_n)$
$\Longrightarrow$
$r'(v_1, v_2, \ldots, v_n)$
$s'(w_1, w_2, \ldots, w_n)$

This rule combine the information in two given terms to produce a pair of simpler terms.

6.9 PAIRCOMB

$r(v_1, 0, \ldots, v_n)$
$s(0, v_2, \ldots, v_n)$
$\Longrightarrow$
$t(v_1, v_2, \ldots, v_n)$

This rule combines the information in two given terms to produce a single resulting term. It decreases the total variable usage count, and decreases the number of zeros occurring in terms. This rule may eliminate some variables from the system, which in turn will reduce the fanout of the dilemma rule.

6.10 FACTOR

$r(v_1, \ldots, v_i, v_{i+1}, \ldots, v_n)$
$\Longrightarrow$
$s(0, \ldots, 0, v_0, \ldots, v_i)$
$t(0, \ldots, 0, v_{i+1}, \ldots, v_n)$

if the given relation can be partitioned into two relations over independent sets of variables.

This rule does not increase the variable usage count. It does, however, increase the number of terms (but not neccesarily the space required to represent them), but this is offset by an increase in the applicability of PAIRCOMB.

7 Remarks

Whenever a dyadic term $r(a, b)$ is generated, it is subtracted from all other terms, since the other terms no longer need to hold this information. By this policy we can ensure that no individual term in the *Rn* set will imply a dyadic term.

This may not always be a good idea, but seems ok provided we never need any inference rules with three antecedents.

New dyadic terms thus only arise from considering pairs of terms in *Rn*, but once found it may immediately generate a cascade of others. These are used to simplify the terms as much as possible before resorting to the pairwise rules.

Terms are deleted only when they are known to be satisfied under all value assignments compatible with the current set of constraints. For example, if the terms $\{x = y\}$,$\{x = 1\}$ and $\{y = 1\}$ are present, the term $\{x = y\}$ can be deleted.

8 Aside

This whole mechanism reminds me of chemistry. The terms are molecules of a mixture of gases, and the variables are atoms. Molecules may spontaneously decompose into simpler molecules, or pairs of them may react to form larger molecules, or possibly multiple fragments. Unlike in real chemistry, if two identical molecules collide, one of them is annihilated. The whole process end with either an explosion or complete annihilation.

9 The Algorithm

```
INIT:           Rn <- input terms
                R2 <- empty
```

```
LOOP:          Apply the deductions and simplification
               rules to Rn and R2 untile no further change.

               for d = 1,2,... <apply dilemma rule depth d>.
```

10 Implementation

10.1 The R2 structure

Use a vector (varmap, say) to represent the simple mappings to 0 and 1, and the equality/inequality mappings.

```
  varmap!x = -1              Map x to 1
  varmap!x =  1              Map x to 1
  varmap!x =  0              Nothing known about x
  varmap!x =  y      (x>y)   Map x to y.
  varmap!x = -y      (x>y)   Map x to ~y.
```

The implication items in $R2$ ($x \to y, x \to \bar{y}, \bar{x} \to y$ and $\bar{x} \to \bar{y}$) are held in a another structure. The addition of one implication to this structure may cause an inconsistency, or may allow some of the implication information to be stored in varmap. Both need to be detected efficiently.

. . . Give design details . . .

What about using an ROBDD to represent $R2$?

10.2 The R8 structure

The terms are represented by 16 32-bit words, the first 8 represent the relation and the remaining 8 the variables used in the relation. These are all non negative. The terms are canoicalised by removing duplicate occurences of variables, removing ignored variables, and then sorting them into increasing numerical order. identical terms are now easy to located. Application of the unit rules are easy. Combination rules require a search for all pairs of terms with at least one variable in common. Such pairs, though numerous, will be relatively rare in proctice, and easy to find.

. . . Give details of the search tricks...

11 Results

Test/Relation/Mapping	Initial term/var Counts	Term/var counts after processing depth n				
		0	1	2	3	4
C6288/BoolOp/EqNe	6288/100	Y	-	-	-	-
C6288/BoolOp/R2	6288/100	Y	-	-	-	-
C6288/R3/EqNe	6288/100	Y	-	-	-	-
C6288/R3/R2	6288/100	Y	-	-	-	-
C6288/R4/EqNe	6288/100	Y	-	-	-	-
C6288/R4/R2	6288/100	Y	-	-	-	-
C6288/R5/EqNe	6288/100	Y	-	-	-	-
C6288/R5/R2	6288/100	Y	-	-	-	-
C6288/R6/EqNe	6288/100	Y	-	-	-	-
C6288/R6/R2	6288/100	Y	-	-	-	-
C6288/R7/EqNe	6288/100	Y	-	-	-	-
C6288/R7/R2	6288/100	Y	-	-	-	-
C6288/R8/EqNe	6288/100	Y	-	-	-	-
C6288/R8/R2	6288/100	Y	-	-	-	-
C7552/BoolOp/EqNe	7552/100	7123/68	6087/57	5611/45	4122/34	Y
C7552/BoolOp/R2	7552/100	Y	-	-	-	-
C7552/R3/EqNe	7552/100	Y	-	-	-	-
C7552/R3/R2	7552/100	Y	-	-	-	-
C7552/R4/EqNe	7552/100	Y	-	-	-	-
C7552/R4/R2	7552/100	Y	-	-	-	-
C7552/R5/EqNe	7552/100	Y	-	-	-	-
C7552/R5/R2	7552/100	Y	-	-	-	-
C7552/R6/EqNe	7552/100	Y	-	-	-	-
C7552/R6/R2	7552/100	Y	-	-	-	-
C7552/R7/EqNe	7552/100	Y	-	-	-	-
C7552/R7/R2	7552/100	Y	-	-	-	-
C7552/R8/EqNe	7552/100	Y	-	-	-	-
C7552/R8/R2	7552/100	Y	-	-	-	-

12 Conclusion

These results give a quantitative measure of the advantages of using multiadic relations over BoolOp relations to represent primary set of terms the algorithm works on, and also the effect of using $R2$ relations to represent the variable mapping over the more restrictive $EqNe$ relations.

The figures should be read with care since they are dependent on the order in which the inference rules are applied. They are not, however, dependent on the order in which variables are selected when applying the dilemma rules, since the tables only contain information about the number of terms remaining at the end of each recursion level.

What the tables clearly show is that the larger the relations used the smaller the number of terms and variables that have to be processed and more importantly the reduction in recurion level required. The choice of relation size is a compromise. As the arity of the relations increase the size of each term increases, prticularly the size of the relation bit pattern since this grows exponentially. A sensible compromise may well be 8 when 256 bits are require to represent the relation which equals the space required to represent the eight variable of the term. The larger the relation bit pattern to more paralleism is possible in the search for applicable inference rules, and this efficiency gain is somewhat assisted by the reduction in the number of terms and the number of variables. The efficiency of processing the variable mapping information is improved by the reduction in number of variables.

Even with this rather straighforward and general implementation using an unoptimising MCPL compiler the executions times are competitive with rival systems

13 Final speculative remarks

In many ways all tautology checking algorithms are similar. There all take some real life problem and convert it into a form that is essentially a Boolean expression. The expression could be closely related to a digital circuit or it may have been derived from some sub-problem encountered in the course of theorem proving. The form of the expression is sometimes canonicalized by restricting which Boolean operators it contains and where they can be placed. One such is conjunctive normal form, but this may be misguided since the resulting expression can be exponentially larger than other representations. However, conjunctive normal form is the basis for the so called SAT methods that have been extensively studied. Methods in which the expression closely follows the form of the input data are sometimes called structured methods, but when sophisticated heuristics are used during checking the distinction becomes blurred.

Whatever form is chosen, it is essentially the conjunction of a set of terms. This is immediately the case in conjunctive normal form, but even an arbitrary Boolen expression can be viewed in this way since, when represented as an abstract syntax tree, every node imposes a relation on the values of nearby quantities. The expression, in whatever form, will contain propositional variables and it is the task of the checker to discover whether or not there is an assignment that will cause all terms in the conjunctive set to be satisfied.

All methods resort to simplification and deductive techniques to reduce the problem to one with an obvious result. Simplication may involve the removal of identical terms, and deductions may discover necessary setting of some of the variables or relationships between pairs of them. But all methods have to resort eventually to some branch and bound technique, which is essentially an exhaustive search over a tree where each node is a decision point and its edges correspond to the different possible choices that can be made. In many methods, a decision point corresponds to the setting of a particular variable to 1 or to 0. After making a decision a number of simplifications and deductions can usually be made before reaching the next decision point. Methods differ in whether they use depth first, breadth first or some combination of the two when exploring the tree, and they differ in the choices they place at the decision points. In many methods, these may be straighforward variable assignments, but the order in which they are selected is critical, and sophisticated heuristics are sometimes used.

References

[KS97] W. Kunz and D. Stoffel. *Reasoning in Boolean Networks.* Kluwer Academic Publishers, 1997.