A Tautology Checker loosely related to Stålmarck's Algorithm

by

Martin Richards

mr@cl.cam.ac.uk

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

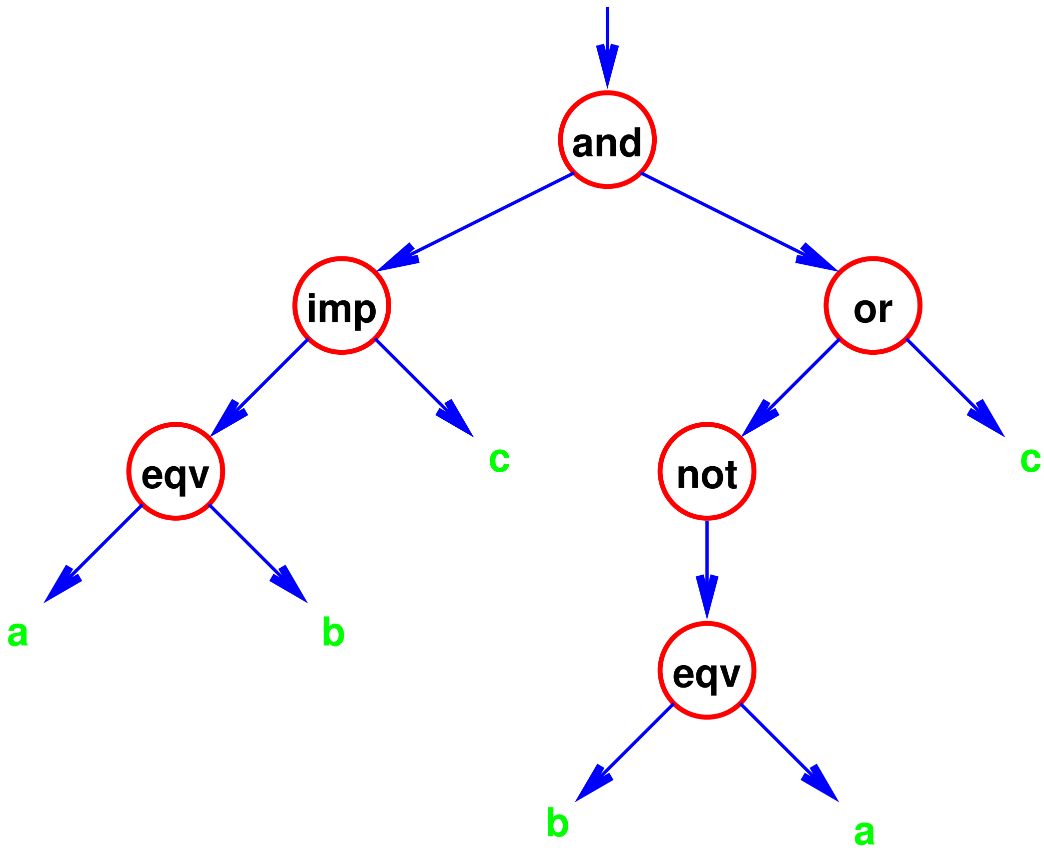
University Computer Laboratory
New Museum Site
Pembroke Street
Cambridge, CB2 3QG

Introduction

Consider the propositional expression:

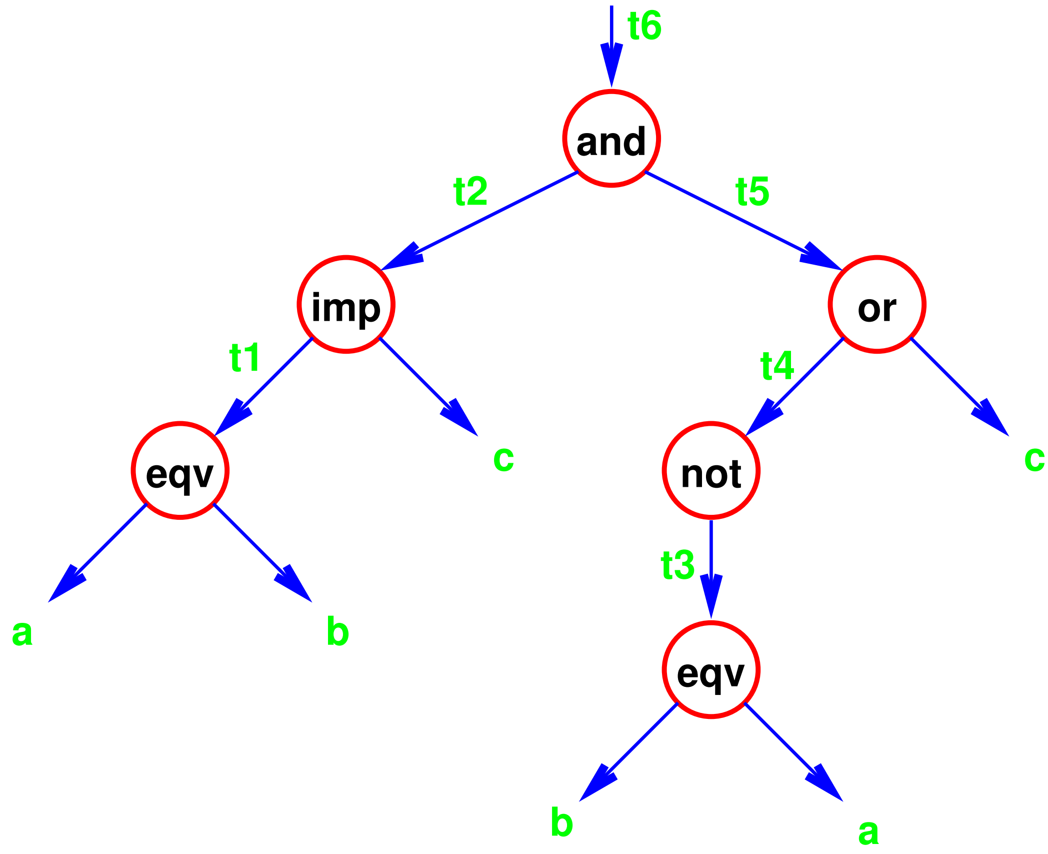
$$((a \equiv b) \rightarrow c) \wedge (\neg(b \equiv a) \vee c)$$

The corresponding tree is:



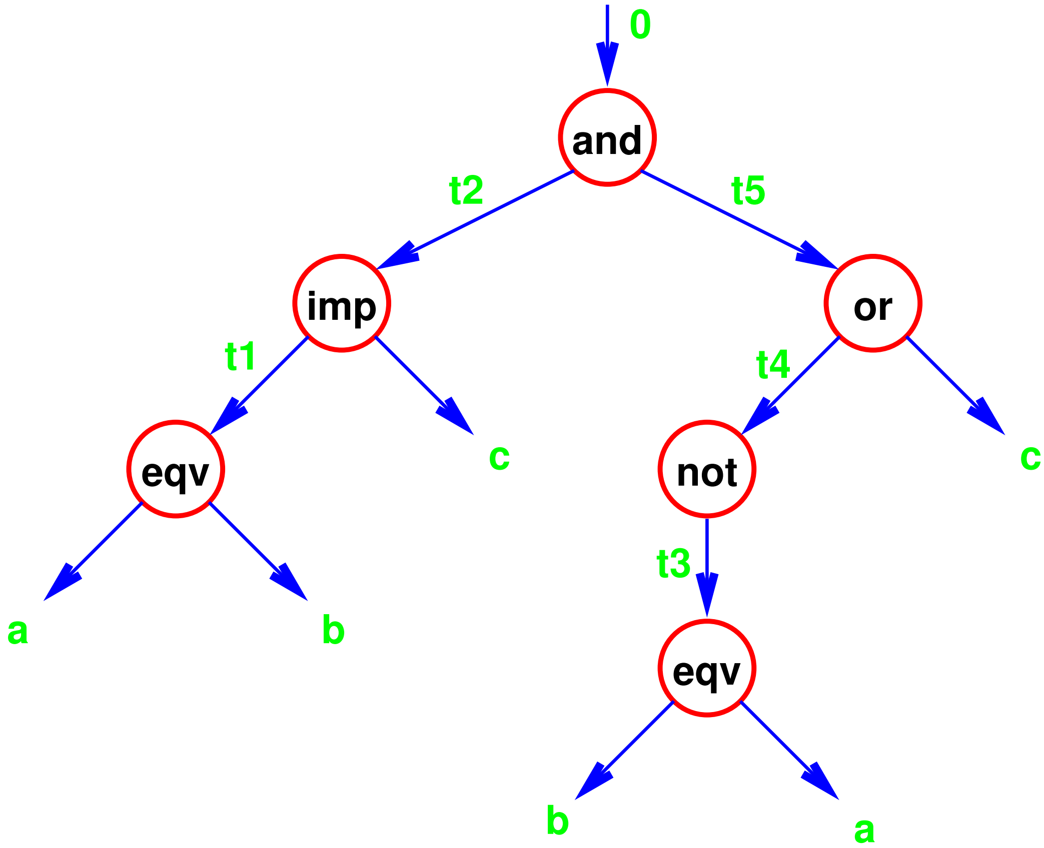
Label Edges

Invent variable names for the unlabelled edges:



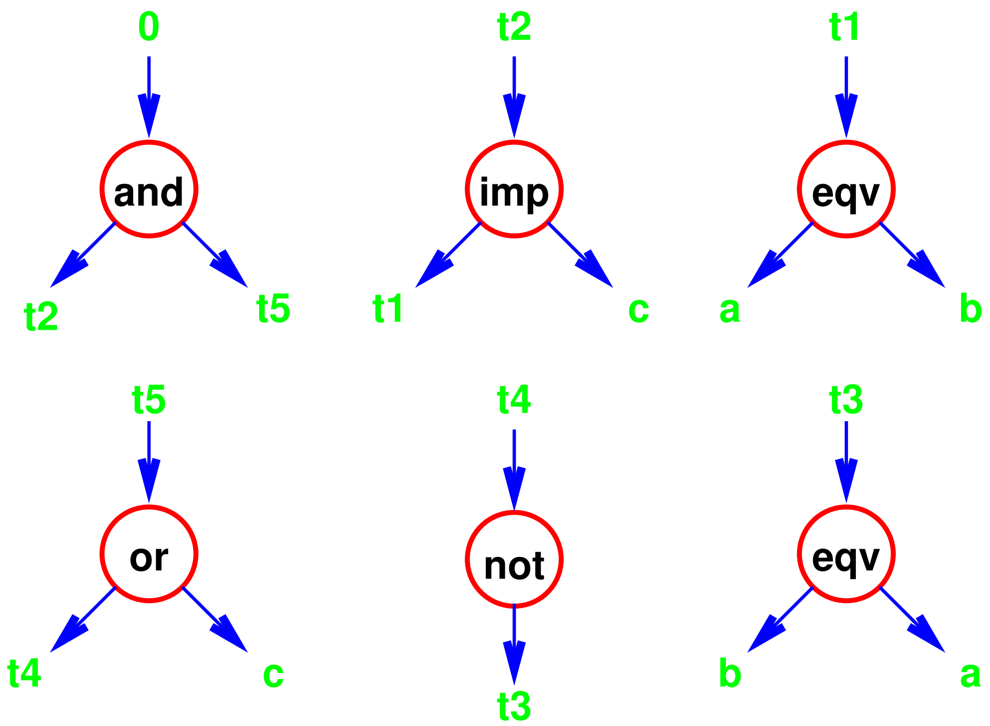
Set Root to Zero

Set the root variable to zero (false) and see if this leads to an inconsistency.



Set of Terms

The tree corresponds to the following set of terms.



Inference Rules

The terms are modified and possibly eliminated by the application of inference rules which sometimes yield variable mapping information.

Mapping information

Variable mapping information is a set of items of the form:

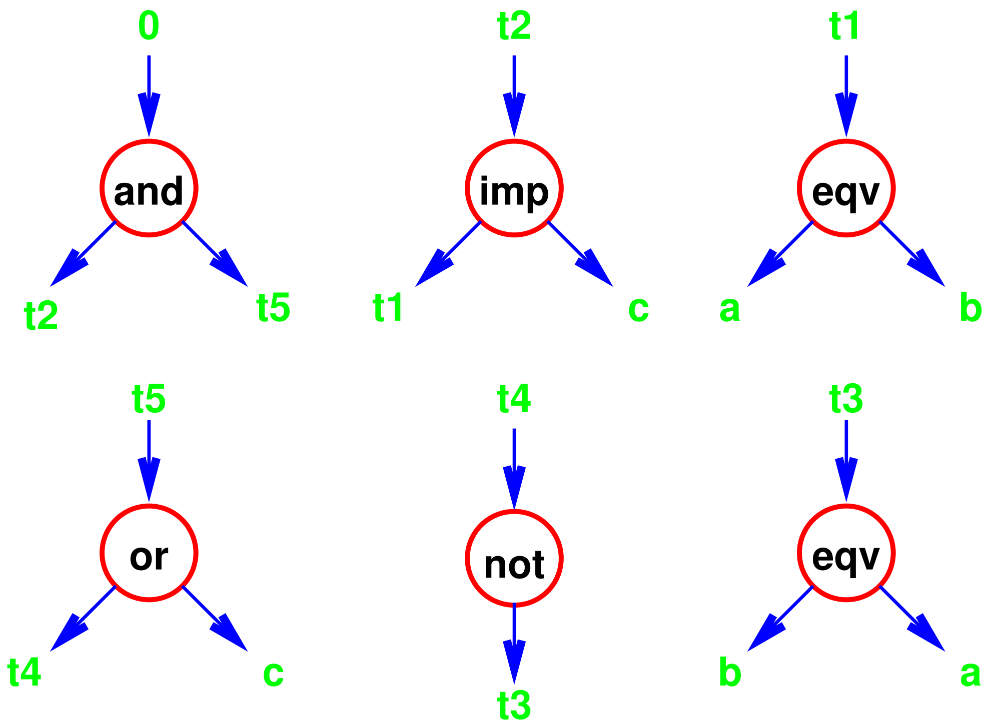
$$x = 0$$

$$x = 1$$

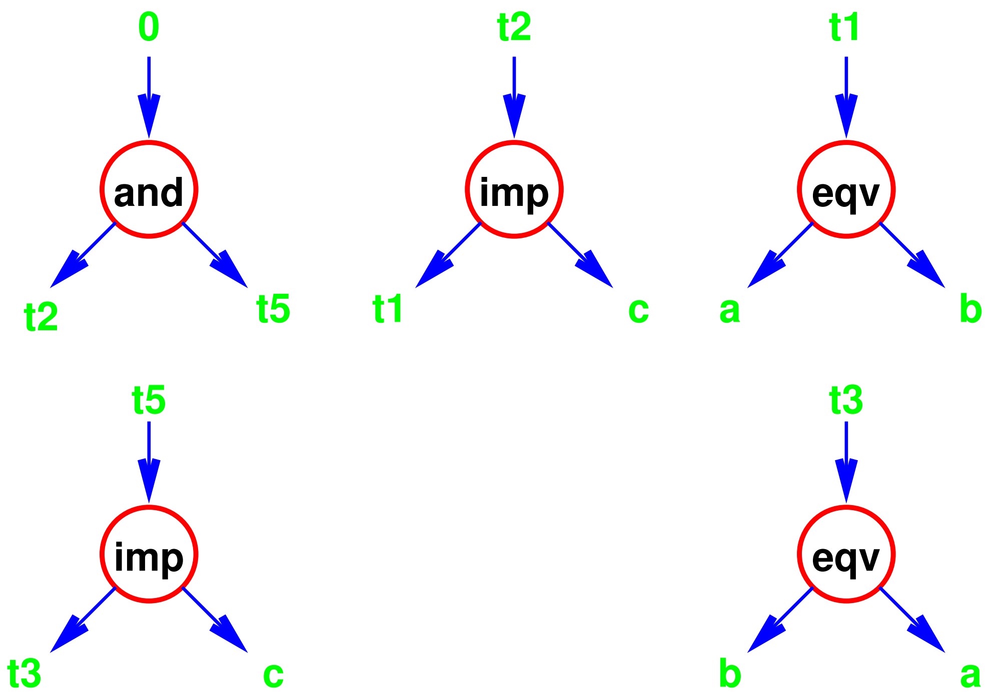
$$x = y$$

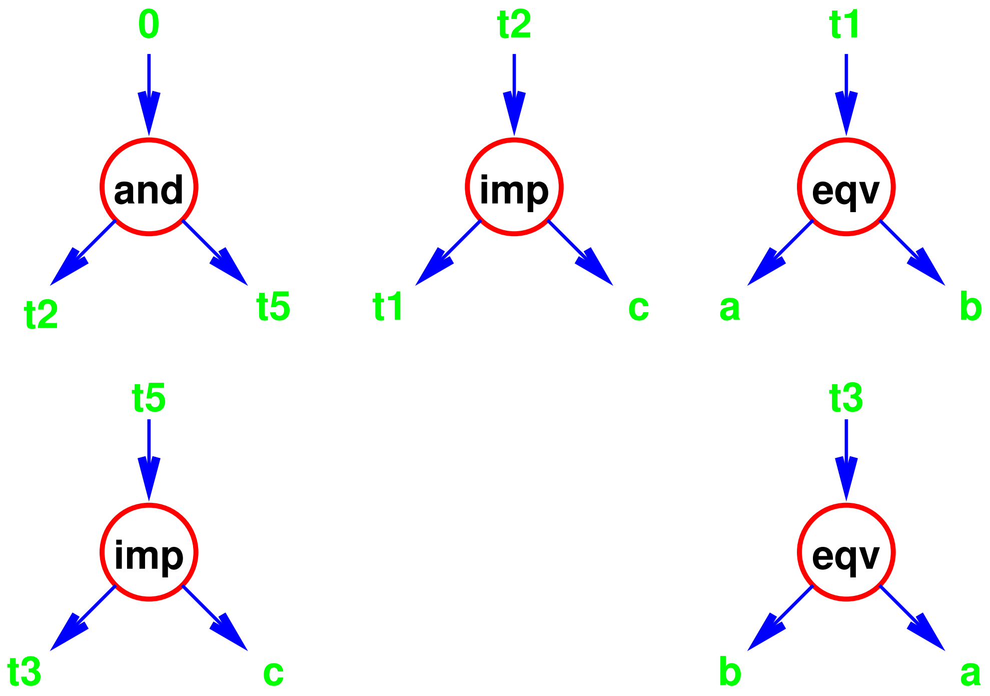
or $$x = \bar{y}$$

where x and y are variables.

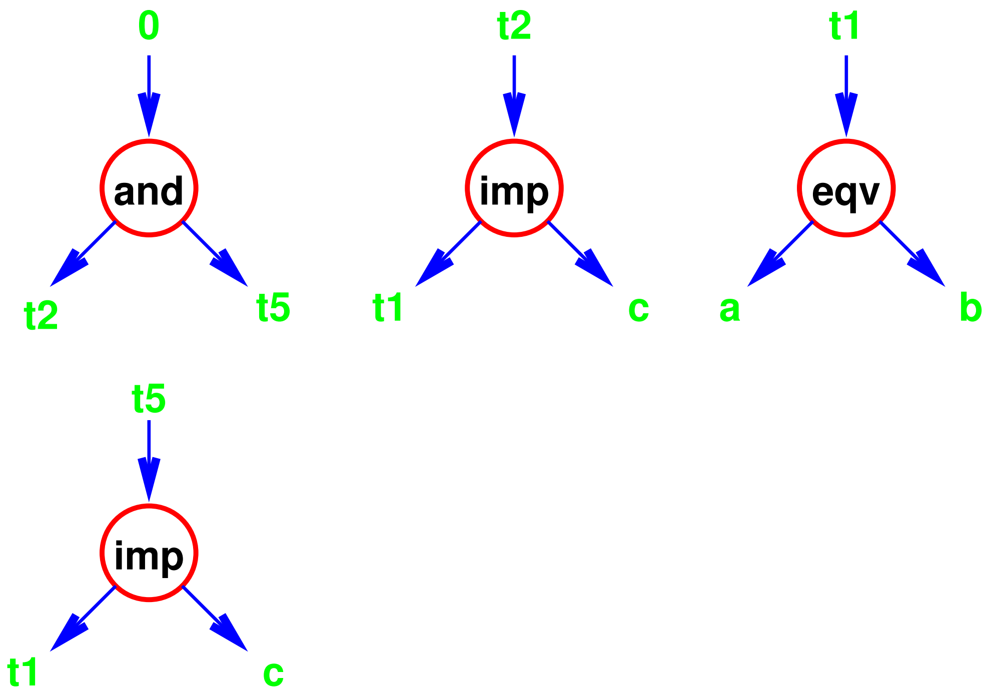


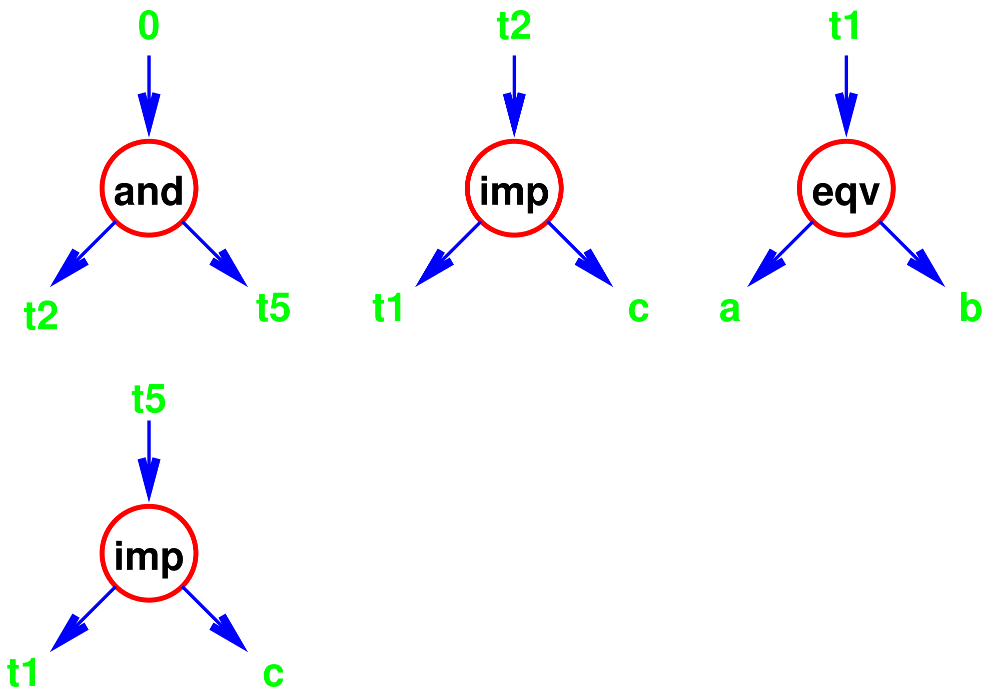
Mapping: $t_4 = \bar{t}_3$



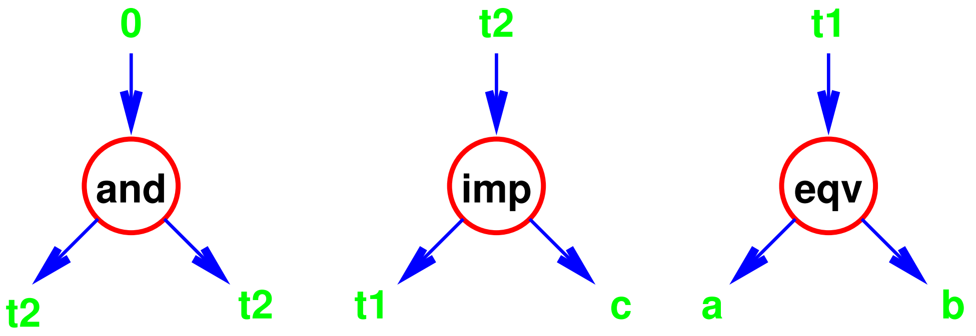


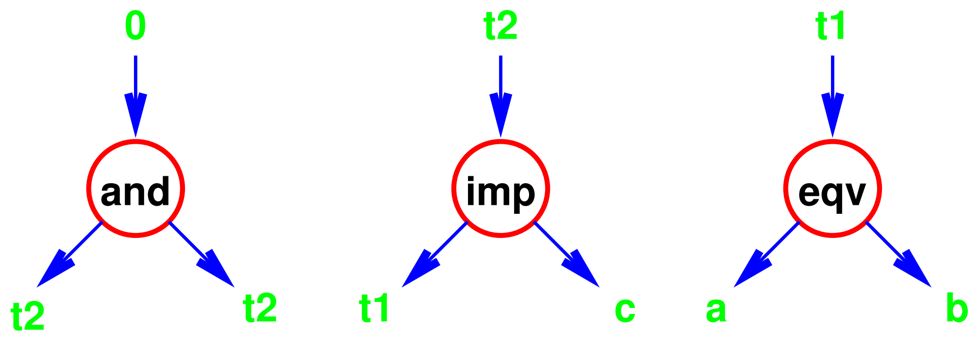
Mapping: $t_4 = \bar{t}_3, \quad t_3 = t_1$





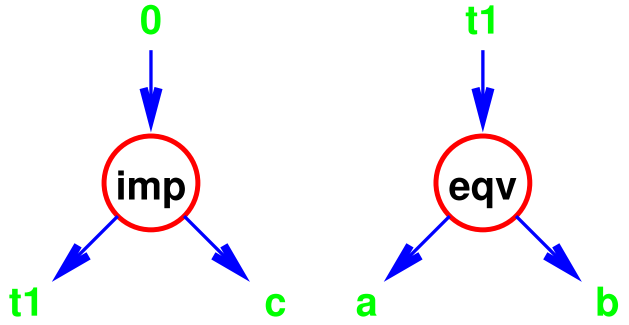
Mapping: $t_4 = \bar{t}_3, \quad t_3 = t_1, \quad t_5 = t_2$

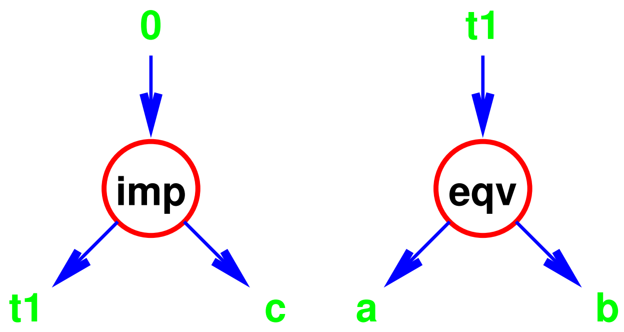




Mapping:

$$t_4 = \bar{t}_3, \quad t_3 = t_1, \quad t_5 = t_2, \quad t_2 = 0$$

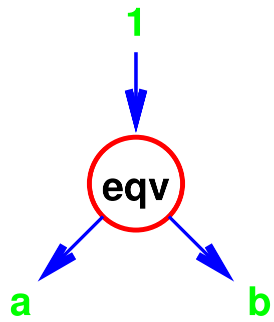


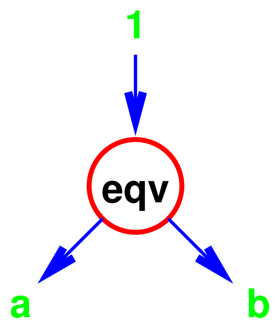


Mapping:

$t_4 = \bar{t}_3, \quad t_3 = t_1, \quad t_5 = t_2,$

$t_2 = 0, \quad t_1 = 1, \quad c = 0$





Mapping:

$t_4 = \bar{t}_3, \quad t_3 = t_1, \quad t_5 = t_2,$

$t_2 = 0, \quad t_1 = 1, \quad c = 0, \quad a = b$

There are now no terms left. An inconsistency has not been found, so the original expression was not a tautology.

These direct derivations can be done in linear time.

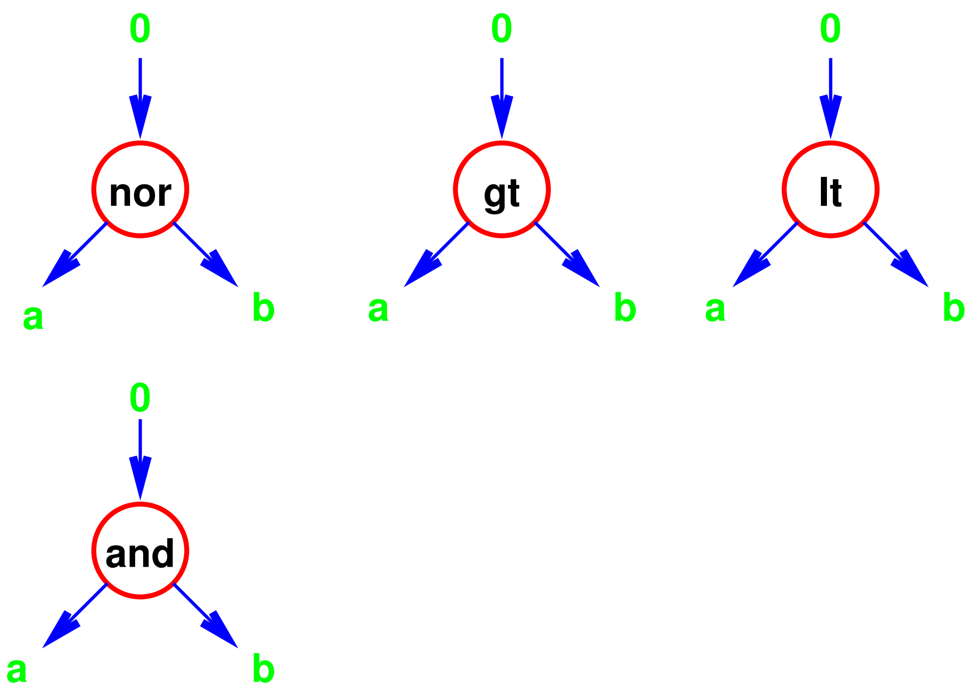
Unfortunately

We cannot, in general, expect direct derivations to solve the problem, (since tautology checking has exponential cost).

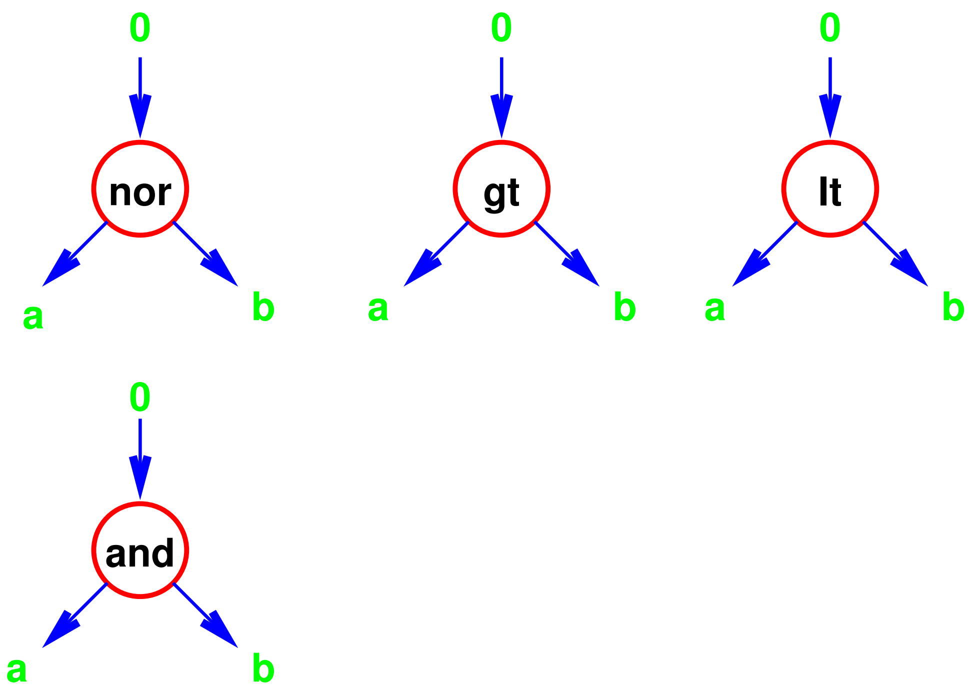
If we apply the technique to:

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

we obtain:



We can easily see that



has no solution since

first term implies:	$ab =$		01,	10,	11
second term implies:	$ab =$	00,	01,		11
third term implies:	$ab =$	00,		10,	11
fourth term implies:	$ab =$	00,	01,	10	

The Dilemma Rule

1. Select a variable, v_i, say, that has not yet been assigned a value.
2. Set it to 0 and 1 in turn and apply the direct inferences until convergence, yielding variable mappings M_0 and M_1, respectively.
3. If both settings lead to inconsistencies then the terms cannot be satisfied.
4. If just one setting leads to an inconsistency, return the other mapping.
5. If neither setting leads to an inconsistency, return the intesection of M_0 and M_1, i.e. the set of mapping items that occur in both M_0 and M_1.
6. Repeat for all other variables.

7. Repeat the process using pairs of variables, $v_i v_j$, setting them to all possible values (00, 01, 10 and 11), taking the intersection of the four mappings produced.

8. Repeat similarly with all sets of 3, 4,... n variables until either no terms remain or an inconsistency is found.

When the algorithm is considering k variables, it is said to be at recursion depth k.

Hopefully, the problem will be solved before the recursion depth gets too large.

The problem will certainly be solved at recursion depth n, if not before.

Any strategy to reduce the recursion depth is valuable.

Mapping Intersection

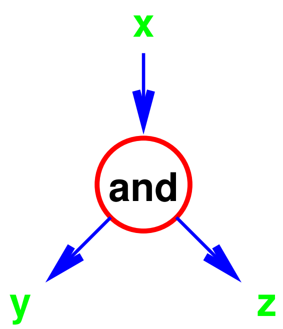
Suppose $M_0 = \{v_1 = 0, v_2 = 1, v_2 = \bar{v}_4, v_5 = v_6\}$
and $M_1 = \{v_1 = 0, v_2 = 0, v_3 = v_5, v_5 = v_6\}$

then the intersection is

$$\{v_1 = 0, v_5 = v_6\}$$

Using Triadic Relations

The terms so far considered represent the relation between the operands and result of dyadic Boolean operators. For example,



says that the only valid settings of xyz are:

000, 001, 010 or 111

There are just 16 dyadic Boolean operators, but there are 256 relations over 3 Boolean variables.

We extend Stååmarck's Algorithm to use such relations in the term set.

Eight Bit Representation

The relation $R(x, y, z)$ can be represented using a bit pattern of length 8, as follows:

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	The bits

$$h = 1 \Leftrightarrow 000 \epsilon R$$

$$g = 1 \Leftrightarrow 001 \epsilon R$$

$$f = 1 \Leftrightarrow 010 \epsilon R$$

$$e = 1 \Leftrightarrow 011 \epsilon R$$

$$d = 1 \Leftrightarrow 100 \epsilon R$$

$$c = 1 \Leftrightarrow 101 \epsilon R$$

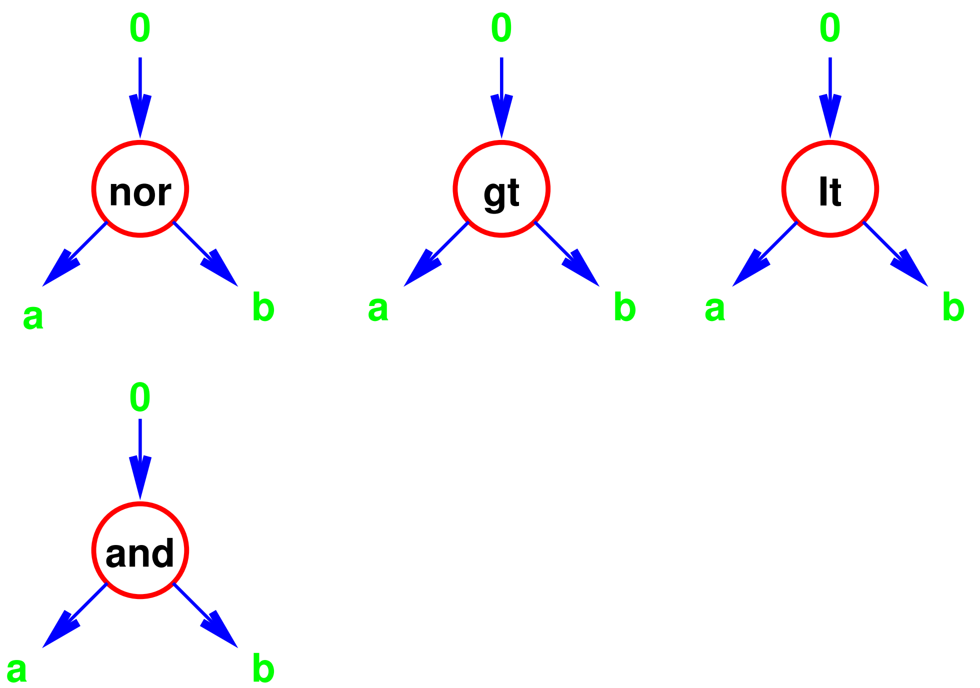
$$b = 1 \Leftrightarrow 110 \epsilon R$$

$$a = 1 \Leftrightarrow 111 \epsilon R$$

Notation

We will use *Rabcdefgh* to identify the relation with bit pattern *abcdefgh*.

The terms in the previous example:



can be written as:

$$R00011110(0,a,b) \qquad R01001011(0,a,b)$$
$$R00101101(0,a,b) \qquad R10000111(0,a,b)$$

New Inferences

Many more inference rules are now available. We can, for instance, combine $R00101101(0, a, b)$ and $R10000111(0, a, b)$ to yield $R00000101(0, a, b)$.

Relation	Triplets							
R00101101			101		011	010		000
R10000111	111					010	001	000
R00000101						010		000

Derivation

The terms in the previous example are:

$$R00011110(0, a, b)$$
$$R01001011(0, a, b)$$
$$R00101101(0, a, b)$$
$$R10000111(0, a, b)$$

Combining terms 3 and 4 gives:

$$R00011110(0, a, b)$$
$$R01001011(0, a, b)$$
$$R00000101(0, a, b)$$

Combining terms 2 and 3 gives:

$$R00011110(0, a, b)$$
$$R00000001(0, a, b)$$

Combining these two terms gives:

$$R00000000(0, a, b)$$

This is a term that cannot be satisfied, so an inconsistency has been found.

Observations

Having 256 relations may seem to be a disadvantage, but there are compensations.

- As we have seen, the recursion depth may be reduced.
- Terms can be put into canonical form to allow easy elimination of equivalent terms.
- Many more inferences are possible.
- Inferences can be done by table lookup or by simple bit pattern operations.

Canonicalisation

A term can be put into canonical form 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 $R00001011(0, y, z)$, since the upper and lower 4 bits of R are equal.

2. 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 $R10011001$.

3. Replace $R(1, y, z)$ by the equivalent $R'(0, y, z)$. For example, replace $R00110101(1, y, z)$ by $R00000011(0, y, z)$, i.e. right shift by 4.

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 canonical term will have one of the following forms:

$$Rabcdefgh(x, y, z)$$

$$R0000abcd(0, y, z)$$

$$R000000ab(0, 0, z)$$

$$R0000000a(0, 0, 0)$$

Deducing Mappings

Given a term $Rabcdefgh(x, y, z)$, we may be able to deduce new mapping information. For instance, if $abcd = 0000$ then $x = 0$.

By similar means, we can deduce whether $x = 1$, $x = y$ or $x = \bar{y}$.

Term	Deduction
$R0000efgh(x, y, z)$	$x = 0$
$Rabcd0000(x, y, z)$	$x = 1$
$Ra00de00h(x, y, z)$	$x = y$
$R0bc00fg0(x, y, z)$	$x = \bar{y}$
$\ldots$	

There are 12 such mappings.

They can be deduced by simple masking operations or by table lookup.

Pairwise Inferences

As we have already seen, if we have two terms $R(x, y, z)$ and $S(x, y, z)$ referring to the same variables, then they can be replaced by the single term $(R \wedge S)(x, y, z)$.

Pairwise Inferences

If we have two terms $R(a, x, y)$ and $S(b, x, y)$ with two variables in common then we can sometimes simplify R and/or S, and we can sometimes deduce a relation between a and b. For example,

Term	*Triplets*			
$R01010011(a, x, y)$	110	100	001	000
$R10101100(b, x, y)$	111	101	011	010

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

$R01000010(a, x, y)$	110	001
$R00100100(b, x, y)$	101	010

Pairwise Inferences

$$R01000010(a, x, y) \quad 110 \quad 001$$

$$R00100100(b, x, y) \quad 101 \quad 010$$

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 operations on the relation bit patterns.

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

Using Larger Relations

We have seen how the mechanism works with relations over three Boolean variables.

What about relations over 4, 5,. . . n variables?

The length of the relation bit pattern is 2^n and the number of variable mapping items is $n(n+1)$.

When $n = 4$, 16-bits are required to represent the relation and there are 20 mapping items.

When $n = 8$, the relation takes eight 32-bit word to represent, which is typically equal to the space require to represent the eight variables. Such a term can thus be represented in 16 32-bit words.

A term over 6 variables would require 8 words.

$n = 8$ is probably a good compromise.

Notation

We will use Rn to denote the set of terms containing general relations over n variables.

A typical element of Rn is $R(v_1, \ldots v_n)$.

$BinOp$ is the subset of $R3$ in which all terms are restricted to relations corresponding to the 16 dyadic Boolean operators.

Stålmarck's variable mapping can be thought of as the subset ($EqNe$) of $R2$, restricted to 12 of the 16 possible relations over 2 Booleans. The four that are omitted involve implication, possibly combined with negation. We will call this subset of $R2$: Imp.

$R2$ Relations

Relation	xy Pairs				Condition	
R0000					$False$	
R0001				00	$x = 0$	$y = 0$
R0010			01		$x = 0$	$y = 1$
R0011			01	00	$x = 0$	
R0100		10			$x = 1$	$y = 0$
R0101		10		00	$y = 0$	
R0110		10	01		$x = \bar{y}$	
R0111		10	01	00	$x \rightarrow \bar{y}$	
R1000	11				$x = 1$	$y = 1$
R1001	11			00	$x = y$	
R1010	11		01		$y = 1$	
R1011	11		01	00	$x \rightarrow y$	
R1100	11	10			$x = 1$	
R1101	11	10		00	$\bar{x} \rightarrow \bar{y}$	
R1110	11	10	01		$\bar{x} \rightarrow y$	
R1111	11	10	01	00	$True$	

$R2$

If a circular chain of implications can be found in $R2$ then either all the variable in the chain are equal or an inconsistency is present. For example:

$$x \rightarrow y, \bar{y} \rightarrow z, z \rightarrow \bar{x} \quad \Longrightarrow \quad x = y = \bar{z}$$
$$x \rightarrow y, \bar{y} \rightarrow \bar{z}, z \rightarrow \bar{x} \quad \Longrightarrow \quad Inconsistent$$

Such deductions should be made as soon as they are detectable.

Elements of $EqNe$ correspond to renaming, and can be removed as soon as the renaming is done.

We are thus left with an $R2$ structure containing only terms from Imp that contain no cycles and we need an efficient algorithm to detect cycles when new items are added (to either Imp or $EqNe$). It is well known that $R2$ can be "solved" in linear time.

PERM

$R(v_1, v_2, \ldots, v_n)$

$\Longrightarrow$

$R'(v_i, v_j, \ldots, v_k)$

where $i, j, ...k$ is a permutation of $1, 2, ..., n$

Example

$R10110110(x, y, z)$	111	101	100	010	001
$\Longrightarrow$					
$R10110110(x, z, y)$	111	110	100	010	001

UNDUP

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

Example

$R10110110(x, x, y)$ 111 101 100 010 001

$\Longrightarrow$

$R00100010(x, 0, y)$ 101 001

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

ELIMVAR

$R(v_1, v_2, \ldots, v_n)$

$\Longrightarrow$

$R'(0, v_2, \ldots, v_n)$

if v_1 is not used in any other term

Example

$R00110110(x, y, z)$ 101 100 010 001

$\Longrightarrow$

$R00000111(0, y, z)$ 010 001 000

INDEP

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

Example

$R01100110(x, y, z)$ 110 101 010 001

$\Longrightarrow$

$R00000110(0, y, z)$ 010 001

ONE

$$R(1, v_2, \ldots, v_n)$$

$$\Longrightarrow$$

$$R'(0, v_2, \ldots, v_n)$$

Example

$R01100101(1, y, z)$ 110   101   010   000

$\Longrightarrow$

$R00000110(0, y, z)$ 010   001

ZERO

$$R(0, v_2, \ldots, v_n)$$

$$\Longrightarrow$$

$$R'(0, v_2, \ldots, v_n)$$

Example

$R01100101(x, y, z)$ 110 101 010 000

$\Longrightarrow$

$R00000101(0, y, z)$ 010 000

UNIT

$$R(v_1, v_2, \ldots, v_n)$$

$$\Longrightarrow$$

$$S(v_i, v_j)$$

Example

$R01001111(x, y, z)$	110	011	010	001	000
$\Longrightarrow$					
$R1011(x, y)$	11	01	00		

PAIR2

$$s(a, b)$$

$$R(v_1, v_2, \ldots, v_n)$$

$$\Longrightarrow$$

$$R'(v_1, v_2, \ldots, v_n)$$

Example

$R1011(x, y)$	11	01	00		
$R01111001(x, y, z)$	110	101	100	011	000
$\Longrightarrow$					
$R01001001(x, y, z)$	110	011	000		

PAIRGEN

$$R(v_1, v_2, \ldots, v_n)$$

$$S(w_1, w_2, \ldots, w_n)$$

$$\Longrightarrow$$

$$T(v_i, w_j)$$

Example

$R01101011(a, y, z)$	111	110	011	001	000
$R01110011(b, y, z)$	110	101	100	001	000
$\Longrightarrow$					
$R1011(a, b)$	11	01	00		

PAIRSIMP

$$R(v_1, v_2, \ldots, v_n)$$

$$S(w_1, w_2, \ldots, w_n)$$

$$\Longrightarrow$$

$$R'(v_1, v_2, \ldots, v_n)$$

$$S'(w_1, w_2, \ldots, w_n)$$

Example

$R01101011(a, y, z)$	111	110	011	001	000
$R01110011(b, y, z)$	110	101	100	001	000

$$\Longrightarrow$$

$R01101011(a, y, z)$	110	001	000		
$R01110011(b, y, z)$	110	101	100	001	000

PAIRCOMB

$R(v_1, 0, \ldots, v_n)$

$S(0, v_2, \ldots, v_n)$

$\Longrightarrow$

$T(v_1, v_2, \ldots, v_n)$

Example

$R0000110000001010(a, 0, y, z)$ 1011 1010 0011 0001

$R0000000011100001(0, b, y, z)$ 0111 0110 0101 0000

$\Longrightarrow$

$R1100000000101000(a, b, y, z)$ 1111 1110 0101 0011

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 is only useful when $n > 5$.

It increases the applicability of PAIRCOMB.

Final Remarks

- This approach reduces the number of terms, the number of variables, and the depth and fanout of the recursion.
- This approach increases the number of inference rules and simplifies their application.
- Multi-variate relations allow the representation of higher level properties of a circuit.
- Another form of dilemma rule is possible. Pick a “random” relation over n variables $R11011100(x, y, z)$, say, and its complement $R00100011(x, y, z)$, and use these as the alternatives in the dilemma rule. This is more useful when $n > 3$.

Last Thought

- If multi-variate relations are a good idea, it might be worth using relations over 16 or perhaps 32 variables, representing the relations by OBDDs instead of a bit patterns.

Analogy