J. Symbolic Computation (1989) **7**, 275–293

Boolean Unification - The Story So Far[*]

Ursula Martin[†] and Tobias Nipkow

Department of Computer Science, RHBNC, University of London
Egham, Surrey TW20 0EX, *UK*
and
MIT Laboratory for Computer Science,
545 *Technology Square, Cambridge MA* 02139, *U.S.A.*

1 Introduction

Unification in Boolean rings, or algebras, has recently attracted considerable interest both for its theoretical merits, since it is unitary, and for its practical relevance for manipulating hardware descriptions, for example in Büttner & Simonis (1986). The aim of this paper is to give a comprehensive survey over the techniques and results in the field as far as it has evolved to date. However most of the results have been around for a long time, some of them going back to Boole himself (Boole, 1847). A very good source for much of the mathematics underlying unification in Boolean rings is Rudeanu (1974).

The most important result is that Boolean unification is unitary, that is either an equation has no solution or there is a single most general unifier (in the sequel: *mgu*). We present two different unification algorithms, one due to Boole and the other to Löwenheim. The paper is structured as follows.

In section we present only the very basic facts about Boolean rings, just enough to explain the unification algorithms presented in section . However the algorithms presented in section are parameterized on procedures for simplifying Boolean terms and finding solutions to Boolean equations. In order to solve these two problems section invokes a bit of structure theory of Boolean algebras. This enables us in sections and to derive solutions to the two remaining subproblems of *simplification* and of finding particular solutions. Section provides a rough complexity analysis of the presented algorithms. It turns out that both unification methods have the same complexity which is exponential in the number of constants and variables. Section explains how the results obtained in terms of a particular set of operators (the ring operations $+$ and $*$) may be transferred to a different set of operators.

So far we only dealt with terms over the ring operations and arbitrary constants. In section we extend unification to include free terms with variables as well. The resulting set of formulae are basically those of the unquantified predicate calculus. It is shown that unification in this theory is finitary. This extension is used in section to obtain a semi-decision procedure for the first order predicate calculus.

[*]This research was carried out at the University of Manchester.

[†]The author acknowledges partial support of the United Kingdom SERC under grant GR81718D.

0747–7171/89/030275+19 $03.00/0

2 Boolean Rings

In this section we present the definition of Boolean rings, a few simple consequences and some important examples.

A set B containing elements 0 and 1 is a *Boolean ring* under the operations $+$ and $*$ if for all $x, y, z \in B$ we have

$$\left.\begin{array}{rcl} x+y & = & y+x \\ (x+y)+z & = & x+(y+z) \\ x+0 & = & x \\ 1*x & = & x \\ (x*y)*z & = & x*(y*z) \\ x*(y+z) & = & x*y+x*z \\ (x+y)*z & = & x*z+y*z \\ x*x & = & x \\ x+(-x) & = & 0 \end{array}\right\} E$$

where 0 is the zero element, 1 the unit and $-x$ is the additive inverse of x. It then follows that 0 is also a zero element with respect to $*$, that $*$ is commutative and every element is its own additive inverse, that is

$$\begin{array}{rcl} x*0 & = & 0 \\ x*y & = & y*x \\ x+x & = & 0. \end{array}$$

In the sequel we will repeatedly make use of the identities $x*(1+x)=0$, and $x^k = x$ for $k \neq 0$. We also abbreviate $x+1$ by $\bar{x}$. The Boolean ring with two elements, 0 and 1, will be denoted by **2**. Since we have $x = -x$ in any Boolean ring we may obtain an equivalent structure axiomatised in terms of $+, *, 0, 1$ alone, by omitting the last equation above. This structure is called a Boolean algebra.

These equations for a Boolean algebra have the following equivalent complete set of rewrite rules under associative-commutative rewriting:

$$\left.\begin{array}{rcl} 0+x & \rightarrow & x \\ x+x & \rightarrow & 0 \\ 0*x & \rightarrow & 0 \\ 1*x & \rightarrow & x \\ x*x & \rightarrow & x \\ x*(y+z) & \rightarrow & x*y+x*z. \end{array}\right\} R$$

This set of rules was first given by Hsiang & Dershowitz (1983). The rules R can be obtained from E in a purely mechanical way by using the Knuth-Bendix procedure enhanced with associative-commutative unification as for example implemented in LP (Lescanne, 1986). If associative-commutative matching is used for $+$ and $*$, R provides a decision procedure for equations over free Boolean rings. It rewrites terms into their so-called "polynomial normal form" (Martin & Nipkow, 1989). This process is described in detail in Hsiang (1985) where it is used for theorem proving.

For our purposes there are two important examples of Boolean rings. The power set $\mathcal{P}(S)$ of a set S with n elements forms a Boolean ring with 2^n elements under the operations of symmetric difference $(+)$ and intersection $(*)$, where 1 is S and 0 the empty set. The set of all well-formed formulae of the propositional calculus on a set of n symbols for propositions forms a Boolean ring with 2^n elements under the operations of exclusive or ($+$) and conjunction ($*$), where 1 is *true* and 0 is *false*. In section we shall see that these two are actually isomorphic.

For the rest of the paper let the signature Σ denote the set of operators $\{0, 1, +, *\}$. Given the signature Σ, a set of variables V and a set of constants C, the term algebra $\mathcal{T}(\Sigma, V \cup C)$ is defined in the usual way. Elements $f \in \mathcal{T}(\Sigma, V \cup C)$ are interpreted as functions from $\mathcal{T}(\Sigma, V \cup C)^n$ (where n is the number of variables in f) to $\mathcal{T}(\Sigma, V \cup C)$; application is substitution. We assume that the variables in each term are ordered linearly and write $f(\underline{x})$ to express that f is a function of the variables $\underline{x} = (x_1, \ldots, x_n)$.

Given a Boolean ring $\underline{B}$, its carrier is denoted by B. The interpretation of a term $f(\underline{x}) \in \mathcal{T}(\Sigma, V \cup B)$ is a function $f^{\underline{B}} : B^n \to B$. We write $s =_{\underline{B}} t$ to denote that $s = t$ is valid in $\underline{B}$. In the following we do not always distinguish between a term f and the function $f^{\underline{B}}$.

3 Two Unification Algorithms

Before we start on the actual business of unification, we simplify the problem due to the following observation. As a consequence of the laws of Boolean rings, an equation $s(\underline{x}) = t(\underline{x})$ is equivalent to $s(\underline{x}) + t(\underline{x}) = 0$. Hence any unification problem is equivalent to a matching problem. Therefore we can restrict our attention to equations of the form

$$f(\underline{x}) = 0. \tag{1}$$

For the rest of this paper let $\underline{B}$ be a Boolean ring and let $f(\underline{x}) \in \mathcal{T}(\Sigma, V \cup B)$, $\underline{x} = (x_1, \ldots, x_n)$, be a term with constants in B.

A second simplification can be justified by the following results.

Lemma 1 *Any Boolean algebra generated by a finite set of generators is finite.*

Theorem 1 *Let C be the finite set of constants in $f(\underline{x})$ and let $\underline{D}$ be the (finite) subalgebra of $\underline{B}$ generated by C. Then the following holds:*

1. *(1) has a solution in $\underline{B}$ if and only if it has a solution in $\underline{D}$.*
2. *Any mgu of (1) with respect to $\underline{D}$ is also an mgu with respect to $\underline{B}$.*

Hence we can restrict our attention to unification in the finite subalgebra generated by the constants in an equation. The result is still correct in any superalgebra thereof.

The following terminology for talking about different kinds of solutions to equations is taken from Rudeanu (1974). Let $S = \{\underline{b} \in B^n \mid f(\underline{b}) = 0\}$ be the set of solutions to (1). An equation of the form (1) is called *consistent* if and only if $S \neq \{\}$. A vector of functions $F : B^m \to B^n$ is called a *(parametric) solution* to (1) if and only if $F(B^m) \subseteq S$. F is called a *general (parametric) solution* if and only if $F(B^m) = S$. F is called a *reproductive solution* if and only if $m = n$, F is a parametric solution, and $F(\underline{s}) = \underline{s}$ holds for all $\underline{s} \in S$. Notice that a general solution does not have to be a most general unifier because it is only guaranteed to yield all *ground* solutions. The importance of reproductive solutions stems from the fact that they are most general unifiers: any solution G can be obtained as an instance of a reproductive solution F because $G = \lambda x.F(G(x))$.

We can now present two different methods for computing reproductive solutions, and hence most general unifiers, for Boolean equations.

3.1 Boole's Method

This method is also known as "successive variable elimination" and appeared first in Boole (1847) for the special case of equations in one variable. Later accounts can be found in Schröder (1890) and Rudeanu (1974). It is based on the following equivalence:
Lemma 2 *Equation* (1) *has a solution in the Boolean ring* $\underline{B}$ *iff*

$$f(0, x_2, \ldots, x_n) * f(1, x_2, \ldots, x_n) = 0, \tag{2}$$

has a solution in $\underline{B}$ *and hence, iff*

$$\prod_{\underline{b} \in \mathbf{2}^n} f(\underline{b}) = 0 \tag{3}$$

holds in $\underline{B}$.

This consistency test was first mentioned in Boole (1847). Note that the correctness of this test is only trivial if (1) is an equation over $\mathbf{2}$. Otherwise a product may well be 0 even if none of its factors are.

Boole's method is based on the equivalence of equations (1) and (2) and proceeds in a simple recursive manner by eliminating variables one by one.

Theorem 2 *Let* $G(\underline{y}) : B^{n-1} \to B^{n-1}$, *where* $\underline{y} = (x_2, \ldots, x_n)$, *be a reproductive solution to* (2). *Then*

$$F(\underline{x}) = ((f(0, G(\underline{y})) + f(1, G(\underline{y})) + 1) * x_1 + f(0, G(\underline{y})), G(\underline{y})) \tag{4}$$

is a reproductive solution to (1).

This theorem is the basis of the algorithm in Büttner & Simonis (1986) which integrates Boolean unification into logic programming.

Example 1 Let $f(x, y) = x + y + x * y + a = 0$ be the unification problem in the free Boolean ring generated by a. Eliminating x results in the subproblem $g(y) = f(0, y) * f(1, y) = y + a * y$. Eliminating y from $g(y)$ results in $g(0) * g(1)$ which simplifies to 0. Hence the original problem is consistent. Applying theorem 2, the reproductive solution for $g(y) = 0$ is $G(y) = ((g(0) + g(1) + 1) * y + g(0)) = (a * y)$. Substituting this into the reproductive solution for $f(x, y) = 0$ yields $F(x, y) = ((f(0, a * y) + f(1, a * y) + 1) * x + f(0, a * y), a * y) = (a * x * y + a * y + a, a * y)$.

Below this recursive algorithm is presented in a functional language notation. It also covers the base case where $\underline{x} = ()$.

```
unify($f(\underline{x})$) =
    if $\underline{x} = ()$
    then if $f(\underline{x}) =_B 0$ then () else fail
    else let $G$ = unify($f(0, \underline{y}) * f(1, \underline{y})$)
         in $((f(0, G(\underline{y})) + f(1, G(\underline{y})) + 1) * x_1 + f(0, G(\underline{y})), G(\underline{y}))$
```

Notice that $f(\underline{x}) =_{\underline{B}} 0$ is not a syntactic comparison but a test whether $f(\underline{x}) = 0$ is valid in $\underline{B}$. For free Boolean rings this test can be carried out using the rewriting system R from section . For arbitrary Boolean rings we present a solution in section .

There are two points about this recursive algorithm which we would like to demonstrate by way of examples.

Although it may appear that unifying a term with n variables leads to n recursive procedure calls, that is not necessarily the case. In practice recursion often stops before because variables have been eliminated by simplification. Take the equation

$$f(x, y, z) = x * y + z = 0. \tag{5}$$

Eliminating z results in the equation

$$f(x, y, 0) * f(x, y, 1) = (x * y) * (x * y + 1) = 0 \tag{6}$$

which still contains two variables. However, a few simplifications reduce (6) to the trivial equation $0 = 0$. Hence any values for x and y are solutions and the subproblem (6) has the reproductive solution $F(x, y) = (x, y)$. Substituting this back yields the reproductive solution

$$F(x, y, z) = (x, y, ((x * y) + (x * y + 1) + 1) * z + (x * y)) = (x, y, x * y)$$

to (5) which is in fact only dependent on x and y. If we think about this procedure in terms of substitutions, it just means that once we get down to a trivial equation, none of the remaining variables need to be instantiated. In the above case x and y remain unchanged (apart from possibly renaming them) and only z is instantiated by $x * y$.

The second important point concerns the selection of which variable to eliminate in each step. Eliminating x instead of z in (5) results in the non-trivial subproblem

$$z * (y + z) = 0. \tag{7}$$

If y is chosen next, this leads to the trivial problem $z * z * z * (1 + z) = 0$ which has the solution $F(z) = (z)$. This in turn leads to the reproductive solutions $F(y, z) = (y * z + y + z, z)$ and $F(x, y, z) = ((y * z + y + z + 1) * x + z, y * z + y + z, z)$ to the equations (7) and (5) respectively. This solution is significantly more complex than the one obtained by eliminating z first. In particular it depends on three variables rather than two.

Recent work by Büttner (Büttner, 1988) has shown how to select the variables in order to get a reproductive solution with the minimal number of parameters.

3.2 Löwenheim's Formula

The second algorithm for computing reproductive solutions to Boolean equations is due to Löwenheim (1908). It consists of finding a particular solution which is then substituted into a general formula to yield a reproductive solution. Detailed expositions of this method can be found in Martin & Nipkow (1986, 1989).

Theorem 3 *Let $\underline{b} \in B^n$ be a particular solution to* (1), *i.e.* $f(\underline{b}) = 0$. *Then the vector of functions*

$$F(\underline{x}) = \underline{x} + f(\underline{x}) * (\underline{x} + \underline{b}) \tag{8}$$

is a reproductive solution to (1).

It should be noted that in the definition of F in the above theorem $+$ and $*$ are extended to operate on vectors of functions.

A simple example will illustrate how theorem 3 works. Let

$$a * x + b * y + a = 0 \tag{9}$$

be the equation we want to solve. One solution to (9) is $x = 1, y = 0$. Hence a reproductive solution to (9) is

$$\begin{aligned} F(x,y) &= (x,y) + (a * x + b * y + a) * ((x,y) + (1,0)) \\ &= (x + (a * x + b * y + a)(x + 1), y + (a * x + b * y + a) * y). \end{aligned}$$

Obviously the only "algorithmic" aspect of this method is to find particular solutions. Various algorithms for this problem are discussed in section .

We would like to conclude the discussion of Löwenheim's formula with a comparison with Boole's method.

From Löwenheim's formula it is obvious that the reproductive solution always contains as many parameters (new variables) as the unification problem contains variables. Hence in general it leads to more complex solutions than the ones resulting from variable elimination. The equation $x + y = 0$ shows this quite clearly: Variable elimination yields the two reproductive solutions $F_x(x,y) = (y,y)$ and $F_y(x,y) = (x,x)$, depending on whether x or y is eliminated first. Löwenheim's formula yields the two solutions $F_{00}(x,y) = (x * y, x * y)$ and $F_{11}(x,y) = (x * y + x + y, x * y + x + y)$, depending on whether the particular solution $(0,0)$ or $(1,1)$ is used.

4 The Structure of Boolean Rings

The following theorems about the structure of the variety of Boolean rings all go back to Stone (1936). In its most general form we have the following representation theorem relating any Boolean ring back to **2**.

Theorem 4 *The variety of Boolean rings and the class of subdirect powers of* **2** *coincide up to isomorphism.*

For finite Boolean rings we have the following more concise characterization:

Theorem 5 *The finite Boolean rings and the finite powers of* **2** *coincide up to isomorphism.*

The finite powers of **2** are simply the finite power set algebras where every set is represented by a finite vector of 1s and 0s indicating the presence or absence of a particular element. We shall see in sections and how the representation of finite Boolean algebras as powers of **2** can be used to solve the problems of simplification and finding solutions. However that requires a more constructive connection than the one given in theorem 5. For this reason we introduce the so called *orthogonal basis* and *orthogonal normal form*.

4.1 The Orthogonal Basis

We now define a subset of a Boolean ring called an orthogonal basis, which gives rise to a normal form for the elements.

Definition 1 *A subset $D = \{d_1, \ldots, d_n\}$ of the finite Boolean ring $\underline{B}$ is called an* orthogonal basis *for $\underline{B}$ iff*

1. *D is a* basis *for $\underline{B}$ as a vector space over* $\mathbf{2}$. *This means that each $b \in B$ can be expressed as a linear combination of elements of D,*

$$b = \sum_{i=1}^{n} b_i d_i \tag{10}$$

where $b_i \in \mathbf{2}$, and that the elements of D are linearly independent, that is,

$$0 = \sum_{i=1}^{n} b_i d_i$$

if and only if each $b_i = 0$.

2. *The elements of D are* orthogonal, *that is,*

$$d_i d_j = 0 \; for \; i \neq j.$$

The *orthogonal normal form* of some element $b \in B$ is its unique representation as a linear combination of the basis elements as in (10). We recall that it follows from the definition that if $b \in \underline{B}$ then $bd_i = b_i d_i \in \{0, d_i\}$.

It follows from Stone's theorem that the finite Boolean ring $\underline{B}$ is isomorphic to the power set of a set, and so it must contain a subset of elements which correspond to the singleton sets under this isomorphism. In fact this subset is just the orthogonal basis. In general we have (Martin & Nipkow, 1989):

Theorem 6 *Let $\underline{B}$ be a Boolean ring generated by $C = \{c_1, \ldots, c_n\}$ and for $U \subseteq C$ define*

$$v^U = \prod_{u \in U} u \prod_{w \in C \setminus U} (1 + w).$$

Then the non-zero v^U are all distinct, and form an orthogonal basis of $\underline{B}$. This orthogonal basis is unique.

For the free Boolean ring on a set of generators C this means that the set of all $\tilde{c_1} \cdots \tilde{c_n}$, where $\tilde{c_i}$ is either c_i or $c_i + 1$, is the unique orthogonal basis with 2^n elements. Thus the ring has order 2^{2^n}.

For non-free rings we can use their *presentation* to give a constructive account of their basis vectors. A Boolean ring $\underline{B}$ is often described in terms of a set of generators $C = \{c_1, \ldots, c_n\}$ and relations $w_1 = 0, \ldots, w_k = 0$ where $w_i \in \mathcal{T}(\Sigma, C)$, the term algebra generated by the constants C over the function symbols $\Sigma = \{0, 1, *, +\}$. Formally we write

$$\underline{B} = < c_1, \ldots, c_n \mid w_1, \ldots, w_k > . \tag{11}$$

This means that $\underline{B}$ is isomorphic to the quotient of $\mathcal{T}(\Sigma, C)_E$ (which is $\mathcal{T}(\Sigma, C)$ factored by the equations E, by the subring of $\mathcal{T}(\Sigma, C)_E$ generated by the elements $w_1, \ldots, w_k$. Thus if ϕ is the natural homomorphism from $\mathcal{T}(\Sigma, C)_E$ onto $\underline{B}$, $\underline{B}$ is generated by $\{c_1\phi, \ldots, c_n\phi\}$. In practice we often drop all mention of ϕ and refer to the c_i as elements of $\underline{B}$.

For Boolean rings with a presentation in terms of generators and relations we have the following test for an element being 0:

Lemma 3 *If $\underline{B}$ has a presentation as in* (11) *let*

$$W = \prod_{i=1}^{k}(w_i + 1)$$

Then $p =_{\underline{B}} 0$ holds for any $p \in T(\Sigma, C)$ if and only if $p * W =_E 0$.

Thus we have reduced the 0-test in $\underline{B}$ to a 0-test in the free algebra. The latter can easily be checked using for example the rewriting system R.

In conjunction with theorem 6 we can determine the set of basis vectors as all those v^U such that $v^U * W \neq_E 0$.

Example 2 Let $\underline{B}$ be the Boolean ring on a, b, c subject to $ab + ac + bc = 0$. There are eight elements v^U, of the form $\tilde{a}\tilde{b}\tilde{c}$ where $\tilde{x}$ can be x or $\bar{x}$. Of these,

$$abc =_{\underline{B}} \bar{a}bc =_{\underline{B}} a\bar{b}c =_{\underline{B}} ab\bar{c} =_{\underline{B}} 0,$$

and the rest

$$a\bar{b}\bar{c}, \bar{a}b\bar{c}, \bar{a}\bar{b}c \text{ and } \bar{a}\bar{b}\bar{c}$$

are non-zero and distinct, and form an orthogonal basis for $\underline{B}$. This can be determined easily using lemma 3.

It should be mentioned that Knuth-Bendix completion can be used for the same purpose. Given the equation $ab + ac + bc = 0$ (and the basic system R) it will generate the consequences $ab \to 0$, $ac \to 0$ and $bc \to 0$. The resulting system of rewrite rules can then be used as a decision procedure for the word problem in $\underline{B}$ and hence to test $p =_{\underline{B}} 0$.

We can now use the orthogonal basis to establish the connection between a finite Boolean algebra and its representation as a power of $\mathbf{2}$. Let $\underline{B}$ be a Boolean algebra with orthogonal basis $D = \{d_1, \dots, d_m\}$. Then $\underline{B}$ is isomorphic to $\mathbf{2}^m$ via $\varphi : \underline{B} \to \mathbf{2}^m$ such that

$$\varphi^{-1}((e_1, \dots, e_m)) = \sum_{i=1}^{m} e_i d_i$$

for $(e_1, \dots, e_m) \in \mathbf{2}^m$.

The isomorphism φ itself can be computed as the homomorphic extension of its definition on the generators. Given a generator c_i, $\varphi(c_i) = (e_1, \dots, e_m)$ where

$$e_j = \begin{cases} 0 & \text{if } c_i d_j =_E 0 \\ 1 & \text{if } c_i d_j \neq_E 0 \end{cases}$$

Example 3 In example 2 the orthogonal basis $D = \{a\bar{b}\bar{c}, \bar{a}b\bar{c}, \bar{a}\bar{b}c, \bar{a}\bar{b}\bar{c}\}$. has been determined. Hence the generators a, b and c are mapped into $\mathbf{2}^4$ as follows: $\varphi(a) = (1,0,0,0)$, $\varphi(b) = (0,1,0,0)$ and $\varphi(c) = (0,0,1,0)$.

5 Simplification

Boole's algorithm requires us to decide whether a term equals 0 in an arbitrary Boolean algebra $\underline{B}$. There are basically two approaches.

The first one relies on rewriting techniques. In the simplest case $\underline{B}$ is a free Boolean algebra and system R can be used to test a term against 0. If $\underline{B}$ is given in terms of generators and relations, there is again a choice. One can either try to add all relations to R and obtain a new canonical rewriting system by running the Knuth-Bendix completion procedure. The resulting system can then be used as a decision procedure. It is not clear to us that the completion always succeeds. Alternatively one can use lemma 3 together with system R.

The second approach uses the existence of an orthogonal basis of $\underline{B}$ of cardinality m to translate the problem into $\mathbf{2}^m$. Given a term $t \in \mathcal{T}(\Sigma, C)$, the isomorphism φ (see section) translates $t^{\underline{B}}$ into $\mathbf{2}^m$. This results in m terms $t_i \in \mathcal{T}(\Sigma, \{\})$ such that $t =_{\underline{B}} 0$ if and only if $t_i =_{\mathbf{2}} 0$ for all i. The latter can be solved by simply evaluating t_i in $\mathbf{2}$ using the truth-tables of $+$ and $*$.

Example 4 To test whether the term $a * b * c$ is equal to 0 in the Boolean ring $\underline{B}$ of examples 2 and 3, we apply φ to it which yields $(1,0,0,0) * (0,1,0,0) * (0,0,1,0)$ and hence, by componentwise evaluation, $(0,0,0,0)$. Therefore $a * b * c =_{\underline{B}} 0$ holds.

The image of $a + b$ under φ on the other hand evaluates to $(1,1,0,0)$. Therefore $a + b = 0$ does not hold in $\underline{B}$.

It should be noted that both methods apply equally well to terms with variables which are then interpreted as new free constants.

6 Finding Particular Solutions

We can now tackle the problem left open in the description of Löwenheim's method: finding a particular solution to a Boolean equation. In section we discuss classical algorithms for solving equations in $\mathbf{2}$. In section we reduce the problem of solving equations in an arbitrary Boolean ring to a number of independent problems in $\mathbf{2}$.

6.1 Solutions in 2

Finding particular solutions to Boolean equations is known to be an NP-complete problem (e.g. Garey & Johnson, 1979). Because of its importance for theorem proving, satisfiability tests for propositional formulae have been at the heart of most early mechanical proof procedures, e.g. Davis & Putnam (1960). As a by-product these procedures usually find particular valuations which satisfy the given formula. More recent work in this area can be found in Van Gelder (1984) and Bryant (1986).

Van Gelder presents a satisfiability test of complexity $O(2^{(0.25+\epsilon)*l})$ if l is the length of the formula. Currently this seems to be the best upper bound on the complexity of satisfiability tests.

The work of Bryant is based on the efficient representation of propositional formulae as graphs. Once the formula has been translated into graph form, satisfiability can be tested in time $O(l)$. However, the translation may require exponential time.

Any equation over $\mathbf{2}$ can be expressed in polynomial form, as a sum of products of variables, or 1 plus a sum of products of variables. Then satisfiability is immediate, since any formula whose polynomial form is not 0 is satisfiable. Thus $f(\underline{x}) = 0$ has a solution provided $f(\underline{x})$ is not identical to 1. A solution can be written down in linear time if $f(\underline{x})$ is in polynomial form. If 1 does not appear in $f(\underline{x})$ then set each $x_i = 0$. Otherwise pick a shortest product of variables appearing in $f(\underline{x})$, set each variable in this product to 1 and the rest to 0.

6.2 Solutions in an Arbitrary Boolean Ring

There are two approaches to this problem. One can either reduce it to a number of problems over $\mathbf{2}$ or solve it directly in the given Boolean ring $\underline{B}$.

For the first approach (Martin & Nipkow, 1989) let $\underline{B}$ be a Boolean ring with m basis elements and let (1) be the equation we want to find a solution to (or determine it is unsatisfiable). This is broken up into 3 steps.

1. Translate $f^{\underline{B}} : B^n \rightarrow B$ into m functions $f_i : \mathbf{2}^n \rightarrow \mathbf{2}$. This is easily done with the help of the isomorphism φ defined at the end of section .
2. Solve m *independent* equations $f_i(\underline{x}_i) = 0$ in $\mathbf{2}$ (see section).
3. Combine the solutions (if any) obtained in step 2 and translate them back into $\underline{B}$ using φ^{-1}.

The second approach (Rudeanu, 1976; Martin & Nipkow, 1989) uses a degenerate version of Boole's algorithm: simply instantiate all parameters in the most general unifier with arbitrary values to yield a particular solution. Alternatively one can simplify the algorithm to yield a particular solution directly: if $\underline{b} \in B^{n-1}$ is a solution to $f(0,\underline{y}) * f(1,\underline{y}) = 0$, $(f(0,\underline{b}),\underline{b})$ is a solution to (1).

Example 5 Let $f(x) = ax + bx + b = 0$ be the equation we want to find a solution to in the Boolean ring $\underline{B}$ of example 2.

Using φ as of example 3, the above equation translates to $(1,0,0,0) * x + (0,1,0,0) * x + (0,1,0,0)$ which yields the following set of independent equations:

$$\begin{aligned} 1 * x_1 + 0 * x_1 + 0 &= 0 \\ 0 * x_2 + 1 * x_2 + 1 &= 0 \\ 0 * x_3 + 0 * x_3 + 0 &= 0 \\ 0 * x_4 + 0 * x_4 + 0 &= 0 \end{aligned}$$

A particular solution, for example $(x_1, x_2, x_3, x_4) = (0,1,0,0)$, can easily be found. In the case of $(0,1,0,0)$ we can immediately read off its translation back into $\underline{B}$ as b. In general we will only get its orthogonal normal form $0 * a\bar{b}\bar{c} + 1 * \bar{a}b\bar{c} + 0 * \bar{a}\bar{b}c + 0 * \bar{a}\bar{b}\bar{c}$. It so happens that this expression simplifies to $\bar{a}b\bar{c}$ which (in $\underline{B}$) is equivalent to b.

If we use the degenerate version of Boole's algorithm, we obtain $f(0) * f(1) = ab$ after eliminating x. Using for example lemma 3, it is easy to verify that $ab =_{\underline{B}} 0$. Hence the equation is consistent and as a special solution for x we obtain $f(0) = b$.

There are many variations on the above methods which we shall not explore. For example in Martin & Nipkow (1986, 1989) the formula is brought into polynomial normal form before its translation into $\mathbf{2}^m$. This makes the search for particular solutions trivial (see section) and puts all the work into the normalization process.

7 Complexity

The following complexity analysis of the various algorithms presented does not apply to any of the rewrite based methods. It concentrates on the algorithms which operate by reduction to $\mathbf{2}^m$.

In the sequel let l be the number of symbols in Θ, the formula under consideration, n the number of variables in Θ and k the set of constants in Θ. Also let m be the number of orthogonal basis vectors of the Boolean algebra $\underline{B}$ generated by C, the set of constants in Θ.

7.1 Simplification

Simplifying a formula in $\mathcal{T}(\Sigma, \{\})$ amounts to evaluating it in $\mathbf{2}$ which has complexity $O(l)$. If the formula is in $\mathcal{T}(\Sigma, C)$, it is equivalent to m evaluations in $\mathbf{2}$ and has complexity $O(m * l)$.

7.2 Boole's Method

Both variable elimination and resubstitution of reproductive solutions can be done in time $O(n)$ using suitable structure sharing techniques (e.g. Boyer & Moore, 1972). However, the length of the term constructed by variable elimination is $O(l * 2^n)$. To test this term against 0 requires time $(l * 2^n)$ in $\mathbf{2}$ and $O(m * l * 2^n)$ in $\underline{B}$. Hence Boole's algorithm requires $O(m * l * 2^n)$ steps. To reduce this complexity, a practical implementation requires the judicious application of simplifications every time a variable is eliminated.

7.3 Finding Particular Solutions

The complexity of the best known algorithms for testing the satisfiability and finding a particular solution to Boolean equations in $\mathbf{2}$ (see section) is still exponential, which is not surprising, given that the problem is NP-complete. For simplicity we do therefore assume that its complexity is $O(l * 2^n)$, reflecting the process of trying 2^n possible valuations which each take time $O(l)$ to be tested.

If a solution in an arbitrary Boolean ring $\underline{B}$ is to be found, the reduction to m independent equations yields a complexity of $O(m * l * 2^n)$. If the degenerate version of Boole's algorithm is used to determine a solution the complexity is the same.

7.4 Löwenheim's Method

The complexity of using Löwenheim's formula is determined completely by the complexity of finding a particular solution, which, by section , is $O(m * l * 2^n)$.

We have seen that both Boole's and Löwenheim's method have the same asymptotic complexity $O(m * l * 2^n)$. If we work in the free Boolean ring generated by k constants, $m = 2^k$ and the complexity of unification is $O(l * 2^{k+n})$, i.e. it is exponential in the number of both variables and constants.

8 Different Operators

Quite frequently problems in the realm of Boolean algebra are not formulated in terms of the ring operations 1, + and * but in terms of a different signature Γ. For example $\Gamma = \{0, \bar{\ }, *, \cup\}$, where $\cup$ is union and $\bar{\ }$ complement. As long as Γ is as expressive as Σ, there is no real problem because one can translate in either direction. For the above Γ the translations take the following form:

$$1 = \bar{0}, \quad x + y = x * \bar{y} \cup \bar{x} * y$$

$$\bar{x} = x + 1, \quad x \cup y = x + y + x * y$$

This suggests the naive approach of translating the unification problem into Σ, solving it there and translating it back. The problem with this is the exponential blowup in the size of the formulae which may occur with each translation.

Alternatively one can translate the unification algorithms from Σ into Γ. We show what that means for Boole and Löwenheim.

For Boole's method this requires the translation of the consistency test (2) and of equation (4). Given the above Γ, the consistency test remains unchanged because $*$ is still at our disposal. The translation of equation (4) however has to go through a number of simplifications ($f(0, G(\underline{y})) * f(1, G(\underline{y})) = 0$!) before the following compact formula can be derived:

$$F(\underline{x}) = (\overline{f(1, G(\underline{y}))} * x_1 \cup f(0, G(\underline{y})), G(\underline{y})) \tag{12}$$

For Löwenheim's method we need to translate equation (8). For Γ as above this yields the following formula:

$$F(\underline{x}) = \underline{x} * \overline{f(x)} \cup \underline{b} * f(\underline{x}) \tag{13}$$

For other sets of operators the task is the same although the result of translating Boole's and Löwenheim's formulae may not be as straightforward as in (12) and (13) above.

9 Unification in the Unquantified Predicate Calculus

In this section we describe how to unify terms of the unquantified predicate calculus, viewed as a Boolean ring under exclusive or ($+$) and conjunction. In fact we solve a slightly more general unification problem, which we now describe, and show it to be finitary.

Let Γ be a signature, C a set of constants and X a set of variables, and suppose each contains no elements of Σ. Let $\mathcal{T}(\Gamma, X \cup C) = \mathcal{T}$ be the corresponding term algebra, as defined in for example Siekmann (1984). Let $\mathcal{S} = \mathcal{T} \setminus X$. Now form

$$\mathcal{B}(\Gamma) = \mathcal{T}(\Sigma, \mathcal{S})_E,$$

the free Boolean ring over $\mathcal{S}$. We shall give a unification algorithm for this ring.

Notice that if P is a set of predicate symbols, C a set of constant function symbols, D a set of non constant function symbols and X a set of variable symbols then the set of well formed formulae of the first order predicate calculus on P, C, D and X (see Monk, 1976) corresponds to a subring of the Boolean ring $\mathcal{B}(P \cup D)$, generated by all terms of the form

$$p(t_1, \ldots, t_n)$$

where $p \in P$ and $t_i \in \mathcal{T}(D, X \cup C)$. We denote this subring by $\mathcal{B}_1(P \cup D)$. Finding a unifier involves finding a substitution

$$\sigma : X \to \mathcal{T}(D, X \cup C).$$

In fact we shall see that any unifier arising from our method will be of this form if the original terms correspond to well formed formulae, i.e. lie in $\mathcal{B}_1(P \cup D)$. Hence our generalisation makes no difference.

Suppose that we wish to solve

$$f(\underline{x}) = 0.$$

By multiplying up and using the distributive law we may assume that

$$f(\underline{x}) = \sum_{U \subset \mathcal{S}} b_U x_U,$$

where the sum is over all finite subsets of $\mathcal{S}$, x_U is the product of all the elements of U, each $b_U \in \mathbf{2}$ and only finitely many b_U are not zero.

A solution consists of a unifier

$$\sigma : X \to \mathcal{B}(\Gamma).$$

However, since we only allow unifiers which induce a map from $\mathcal{B}(\Gamma)$ to itself, and since any variable occurs only as a proper subterm of an element of $\mathcal{S}$, it can only be mapped by σ to an element of $\mathcal{T}$. Thus

$$\sigma : X \to \mathcal{T}.$$

Since our equation involves only a finite subset T of $\mathcal{S}$ we may regard it as an equation in $\mathcal{T}(\Sigma, T)_E$, and rewrite it in terms of the orthogonal basis $\{x^U \mid U \subseteq T\}$ as

$$\sum_{U \subseteq T} b^U x^U = 0, \tag{14}$$

where each $b^U \in \mathbf{2}$, and at least one b^U is non-zero.

Now multiplying this equation (14) by x^U for each non-zero b^U, we see that (14) is equivalent to

$$x^U = 0 \quad \text{for each non-zero } b^U. \tag{15}$$

Now

$$x^U = u_1 \ldots u_r(1 + v_1) \ldots (1 + v_s)$$

where u_i and v_j are all distinct elements of $T \subset \mathcal{S}$. Applying a unifier σ only replaces the u_i and v_j by other elements of $\mathcal{S}$, so that

$$x^U \sigma = 0$$

if and only if $u_i\sigma = v_j\sigma$ for some values of i and j.

This gives us our unification procedure. We try to unify the $r.s$ possible pairs u_i, v_j using a unification algorithm in $\mathcal{T}$; either Robinson's unification (Robinson, 1965) or the improved algorithm of Vitter & Simons (1986). If there are no u_i or no v_j, or if none of the pairs can be unified our original equation had no solution. Otherwise, for each unifier that we have found, we do the following. First we use it to simplify the remaining equations in (15). If all the equations simplify to $0 = 0$, we have a unifier for the original equation. Otherwise we repeat the process on one of the simplified equations which is not $0 = 0$. This process terminates, since we only find finitely many unifiers at each stage, and reduce the number of equations in (15) each time. If the original equation has a solution we can either stop the search process when we have found one solution, or continue and find all solutions. There will be a finite set of unifiers that gives all solutions.

Let us consider some examples.

Example 6 To solve

$$p(x)p(a) + p(x)p(b) = 0$$

In terms of an orthogonal basis this becomes

$$p(x)p(a)(1 + p(b)) + p(x)(1 + p(a))p(b) = 0$$

Thus we have to solve the two equations

$$p(x)p(a)(1 + p(b)) = 0$$

and

$$p(x)(1 + p(a))p(b) = 0$$

To solve the first we must try and unify the pairs $p(x), p(b)$ and $p(a), p(b)$. The first gives the most general unifier $\sigma : x \longrightarrow b$; the second pair cannot be unified. Now σ reduces the second equation to

$$(1 + p(a))p(b) = 0$$

which gives rise to one pair of terms $p(a), p(b)$, which cannot be unified. Thus the original equation had no solution.

Example 7 To solve

$$p(x) + p(x)p(b) + p(a)p(b) + p(x)p(a)p(b) = 0.$$

This reduces to three equations

$$\begin{aligned} p(x)(1 + p(a))(1 + p(b)) &= 0 \\ p(x)p(a)(1 + p(b)) &= 0 \\ (1 + p(x))p(a)p(b) &= 0 \end{aligned}$$

The first gives rise to two pairs, $p(x), p(a)$ and $p(x), p(b)$, and two unifiers, $\sigma_1 : x \longrightarrow a$ and $\sigma_2 : x \longrightarrow b$. Under σ_1 the remaining equations become

$$p(a)(1 + p(b)) = 0$$

and

$$0 = 0.$$

The first gives a pair $p(a), p(b)$, which cannot be unified. Under σ_2 both equations reduce to $0 = 0$, so $x \longrightarrow b$ is the most general unifier.

Remark Herold (1986) describes a technique for combining equational theories which, although it sets out to solve a different and more general problem, yields a valid technique in this case. To solve the equation

$$\sum_{U \subseteq S} b_U x_U = 0,$$

take all pairs s_1, s_2 of distinct elements of $\mathcal{S}$ which occur in it and attempt to unify them using the unification algorithm in $\mathcal{T}$. If no pairs can be unified, halt with failure. Otherwise, for each unifier obtained, use it to simplify the equation and then repeat the process. It is easy to see that this method is essentially the same as ours, except that we write our equation in orthogonal form before attempting to unify pairs. This reduces the number of pairs we try to unify, from $n(n-1)/2$ to at most $n^2/4$ at the first pass of the algorithm, when R has n elements.

10 A Semi-Decision Procedure for the First Order Predicate Calculus

In this section we describe a semi-decision procedure for the first order predicate calculus, based on our unification techniques. The reader is referred to chapter 11 of Monk (1976) for definitions.

Theorem 7 *The following steps constitute a semi-decision procedure for the first order predicate calculus. Let P be a formula of the first order predicate calculus. To determine whether P is unsatisfiable*

1. *Form the Skolem expansion*
$$P_S = \forall x_1, \ldots, \forall x_n F(x_1, \ldots, x_n)$$
of P, where $F(x_1, \ldots, x_n) = F$ is quantifier free, and P_S contains no free variables.

2. *Transform F into a Boolean term $B(x_1, \ldots, x_n) = B$ by replacing $p \wedge q$ by pq, $p \vee q$ by $p + q + pq$ and $\neg p$ by $1 + p$.*

3. *Let*
$$B_i = B(x_{in+1}, \ldots, x_{(i+1)n}),$$
for $i \geq 0$, and let
$$B[k] = B_0 . \ldots . B_k.$$
For each $k \geq 0$, determine, using the method of section 6, or otherwise, whether or not $B[k]$ can be unified with 0.

4. *P is unsatisfiable if and only if some $B[k]$ can be unified with 0.*

Note We do not intend to address questions of efficiency here. Of course the equations $B[i]$ are not independent of each other, and information acquired in failing to unify $B[k]$ with 0 can be used in attempting to unify $B[k+1]$ with 0. This will be more efficient than just applying the method of section from scratch for each value of k.

We illustrate the theorem with two examples.

Example 8 Let
$$B(x) = p(x) + p(x)p(a).$$
The substitution $\sigma : x \longrightarrow a$ unifies $B(x)$ with 0, and so
$$\forall x . p(x) \wedge \neg p(a)$$
is unsatisfiable.

Example 9 Let

$$B(x) = p(x) + p(a_1)p(a_2)\dots p(a_n).$$

Then

$$B[0] = B(x_1),$$

and

$$B[n-1] = (p(x_1) + p(a_1)p(a_2)\dots p(a_n))\dots(p(x_n) + p(a_1)p(a_2)\dots p(a_n)).$$

Then the substitution

$$\sigma : x_i \to a_i \ \ (i = 1, \dots, n)$$

is a unifier of $B[n-1]$ with 0, and so

$$(p(x) \vee (p(a_1) \wedge \dots \wedge p(a_n)) \wedge \neg(p(x) \wedge p(a_1) \wedge \dots \wedge p(a_n))$$

is unsatisfiable. (Notice that none of $B[0], \dots, B[n-2]$ can be unified with 0.)

The idea of using unification for first order theorem proving seems to have appeared first in Herbrands thesis Herbrand (1930), chapter 5, section 2, pp. 139-148. He decided which terms to try and unify by associating to each formula an array of signed letters defined by recursion on the variables in the proposition, which generated infinitely many derived propositions. The original proposition was unsatisfiable if and only if one of the derived propositions could be reduced, using unification, to an unsatisfiable formula of the propositional calculus. Robinson (1965) eliminated the complicated notion of the array of signed letters by always working with skolemised formulae; two terms were unified if this would allow the resolution rule.

$$\frac{P \vee Q,\ \neg P \vee R}{Q \vee R}$$

to be applied. Again this gives a semi-decision procedure.

Our method may be regarded as no more than a variant of resolution. The point of the reduction to orthogonal form inside our unification algorithm is that terms are only unified if we can apply the rule $(1+u)u = 0$, that is

$$\frac{P \wedge \neg P}{\cdot}.$$

In our method the different equations $B[k]$ are treated explicitly; in Robinson's method they are hidden.

Another approach to first order theorem proving has been investigated by Hsiang (1985) and Kapur & Narendran (1985). They work in a Boolean ring, as we do, but they do not attempt to unify the $B[k]$ with 0 directly. Instead they assume $B[0]$ is not unsatisfiable, then use unification inside the Knuth Bendix algorithm, attempting to produce directed rules involving Boolean terms which eventually lead to the rule $1 \to 0$, contradicting the assumption.

We now sketch the proof of Theorem 7.

Proof of theorem 7 Our proof depends upon Herbrand's theorem (see Herbrand (1971), chapter 5, section 3, or Monk (1976), theorem 11.42). As we have remarked above, our theorem is similar to that of Herbrand's thesis, chapter 5, section 2, which Herbrand deduced from the result now called Herbrand's Theorem. In the notation above this theorem states that the formula P is unsatisfiable if and only if there is some integer m and $n.m$ variable-free terms $a_1, \ldots, a_{nm}$ such that the conjunction of the m expressions

$$F(a_{1+rn}, \ldots, a_{(r+1)n}) \quad (r = 0, \ldots, m-1)$$

is unsatisfiable in the propositional calculus.

Suppose that P is unsatisfiable. Herbrand's theorem implies that there exist elements $a_1, \ldots, a_{nm}$ with the conjunction of the m expressions

$$F(a_{1+rn}, \ldots, a_{(r+1)n})$$

unsatisfiable. Translating into the Boolean ring, this means that the product of the m expressions

$$B(a_{1+rn}, \ldots, a_{(r+1)n})$$

is equal to zero, and the substitution

$$\sigma : x_i \longrightarrow a_i \quad (i = 1, \ldots, nm)$$

unifies $B[m-1]$ with O.

Conversely suppose that $B[k]$ has been unified with O, by the substitution $\sigma : x_i \longrightarrow t_i$. Replace each variable occuring in the t_i by a variable-free term, to get a substitution $\tau : x_i \longrightarrow a_i$, where each a_i is a variable-free term. Now since $B[k]\tau = O$, the conjunction of the $k+1$ formulae

$$F(a_{in+1}, \ldots, a_{(i+1)n})$$

is unsatisfiable, so by Herbrand's theorem P is unsatisfiable.

11 Conclusion

We would like to conclude by referring to some recent work generalizing the methods presented in this article.

We have dealt with unification of terms over the ring operations and arbitrary constants, and with a special case of adding free function symbols. In Schmidt-Schauß(1988) and Boudet *et. al.* (1988) the general problem of combining boolean unification with unification algorithms for other disjoint theories is solved. Although methods for combining unification algorithms (Yelick, 1987; Herold, 1986; Tidén, 1986) were known before, the syntactic form of the Boolean ring axioms rendered them inapplicable: both non-regular ($x + x = 0$) and collapsing ($x * x = x$) axioms are present.

Recently there have been two generalizations of Boolean unification reported in Nipkow (1988) and Büttner (1988). Nipkow generalizes to primal (or functionally complete) algebras and their varieties. This covers in particular matrix rings over finite fields, Post-algebras and p-rings (rings with $px = 0$ and $x^p = x$ for some prime p). Büttner announces a unification algorithm for Post-algebras based on their lattice structure and presents a unification algorithm for a functionally complete variant of the 4-element Boolean lattice.

References

Boole, G. (1847). The Mathematical Analysis of Logic. Macmillan 1847. Reprinted 1948, B. Blackwell.

Boudet, A., Jouannaud, J.-P., Schmidt-Schauß, M. (1988). Unification in Free Extensions of Boolean Rings and Abelian Groups. *Proc. LICS'88*, 121-131.

Boyer, R.S., Moore, J.S. (1972). The Sharing of Structure in Theorem Proving Programs. *Machine Intelligence* 7, Edinburgh University Press.

Bryant, R.E. (1986). Graph-Based Algorithms for Boolean Function Manipulation, *IEEE Trans. on Computers* 35, 8, 677-691.

Büttner, W. (1988). Unification in Finite Algebras is Unitary (?). *Proc. CADE-9*, LNCS 310, 368-377.

Büttner, W., Simonis, H. (1987). Embedding Boolean Expressions into Logic Programming. *JSC* 4, 191-205.

Davis, M., Putnam, H. (1960). A Computing Procedure for Quantification Theory. *JACM* 7, 201-215.

Garey, M.R., Johnson, D.S. (1979). Computers and Intractability. W.H. Freeman and Company, San Francisco.

Herbrand, J. (1971). Investigations in Proof Theory. *Jacques Herbrand Logical Writings*, W. Goldfarb (ed.), D. Reidel, 44-202.

Herold, A. (1988). Combination of Unification Algorithms. *Proc. CADE-8*, LNCS 230, 450-469.

Hsiang, J. (1985). Refutational Theorem Proving Using Term-Rewriting Systems. *Artificial Intelligence* 25.

Hsiang, J., Dershowitz, N. (1983) Rewrite Rules for Clausal and Non-Clausal Theorem Proving. *Proc. 10th ICALP*, Barcelona, LNCS 154, 431-446.

Kapur, D., Narendran, P. (1985) An equational approach to theorem proving in the first order predicate calculus. *Proc. IJCAI 85*, Los Angeles.

Lescanne, P. (1986). REVE a Rewrite Rule Laboratory. *Proc. CADE-8*, LNCS 230, 696-697.

Löwenheim, L. (1908). Über das Auflösungsproblem im logischen Klassenkalkül. *Sitzungsber. Berl. Math. Gesell.* 7, 89-94.

Martin, U., Nipkow, T. (1986). Unification in Boolean Rings. *Proc. CADE-8*, LNCS 230, 506-513.

Martin, U., Nipkow, T. (1989). Unification in Boolean Rings. *Journal of Automated Reasoning* 4, 381-396.

Monk, J.D. (1976). Mathematical Logic. Springer.

Nipkow, T. (1988). Unification in Primal Algebras. *Proc. CAAP'88*, LNCS 299, 117-131.

Nipkow, T. (1988). Unification in Primal Algebras, Their Powers, and Their Varieties. Submitted for publication.

Robinson, J.A. (1965). A machine oriented logic based on the resolution principle. *JACM* 12, 23-41.

Rudeanu, S. (1974) Boolean Functions and Equations. North-Holland.

Schmidt-Schauß, M (1988). Unification in a Combination of Arbitrary Disjoint Equational Theories. *Proc. CADE-9*, LNCS 310, 378-396.

Schröder, E. (1890): Vorlesungen über die Algebra der Logic. (Leipzig, Vol 1, 1890; Vol 2, 1891, 1905; Vol 3, 1895), Reprint 1966, (Chelsea, Bronx NY).

Siekmann, J. H. (1984). Universal Unification, *Proc. CADE-7*, LNCS 170.

Stone, M.H. (1936). The Theory of Representation for Boolean Algebras. *Trans. Amer. Math. Soc.* 40, 37-111.

Tidén, E. (1986). Unification in Combinations of Collapse-Free Theories with Disjoint Sets of Function Symbols. *Proc. CADE-8*, LNCS 230, 431-449.

Van Gelder, A. (1984). A Satisfiability Test for Non-Clausal Propositional Calculus. *Proc. CADE-7*, LNCS 170.

Vitter, J.S., Simons, R.A. (1986). New Classes for Parallel Complexity: A Study of Unification and Other Complete Problems for $\mathcal{P}$. *IEEE Transactions on Computers*, 403-418.

Yelick, K (1987). Unification in Combinations of Collapse-Free Regular Theories. *JSC* 3, 153-181.