ICOT Technical Memorandom: TM-1008

TM-1008

A Study on Boolean Constraint Solvers

by

S. Menju, K. Sakai, Y. Sato & A. Aiba

February, 1991

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03)3456-3191 ~ 5
Telex ICOT J32964

Institute for New Generation Computer Technology

A Study on Boolean Constraint Solvers

Satoshi Menju, Kô Sakai, Yosuke Sato and Akira Aiba

ICOT Research Center
1-4-28 Mita, Minato-ku, Tokyo 108, JAPAN

Abstract

Some Boolean constraint solving algorithms, for example Boolean unification and a modified Buchberger algorithm, calculate simple formulas representing all solutions for given Boolean constraints. First, we present another algorithm, which is based on a concept similar to that of Boolean unification but uses no extra variables. Second, we report a comparison of these algorithms, namely the modified Buchberger algorithm, Boolean unification, and the new algorithm, in a few examples. This result shows that the new algorithm is the most efficient for many problems in which constraint relations are sparse.

1. Introduction

We are developing a Constraint Logic Programming System called CAL at ICOT [Sakai 89]. The current CAL system solves a Boolean constraint using a modification of the Buchberger algorithm [Sakai 90, Sakai 91]. In this paper, we report another Boolean constraint solver that has the following good properties of Boolean unification [Martin 86, Büttner 87]:

1. It is more efficient for many problems in which constraint relations are sparse.
2. If there is a unique solution, then the solution is calculated. If there are many solutions, then a set of formulas representing all solutions are calculated. In this case, we can easily get a solution by assigning arbitrary values to free variables in the calculated formulas.
3. We can cut down the computation of unimportant variables.

The new solver has several more advantages. For example,

1. Because it uses no extra variables, it is easy to understand the calculated formulas.
2. We can calculate the relations of the designated variables, if any.
3. If we apply a special reduction to the calculated formulas, the reduced formulas take a canonical form of the given constraints.

In the next section, we present the basic notations and properties of Boolean algebras. In Section 3, we discuss the new algorithm for Boolean constraints. In Section 4, we compare the new algorithm with Boolean unification and the modified Buchberger algorithm in a few examples.

2. Preliminaries

We assume that the reader is familiar with Boolean algebras (see [Halmos 63], for example).

For a Boolean algebra $\langle \mathbf{B}, \vee, \wedge, \neg, 0, 1\rangle$, we define two operations $+$ and $\times$ as follows:

$$\begin{aligned} X + Y &\overset{\text{def}}{=} (X \wedge \neg Y) \vee (\neg X \wedge Y), \\ X \times Y &\overset{\text{def}}{=} X \wedge Y. \end{aligned}$$

As usual, we define that the connective of $\times$ is stronger than $+$'s, and omit $\times$ symbols for convenience when there is no confusion. Thus, expression $XY + Z$ is an abbreviation of $(X \times Y) + Z$. Algebraic structure $\langle \mathbf{B}, +, \times, 0, 1\rangle$ forms a commutative ring with unit, which is called a Boolean ring, with the following properties.

$$\begin{aligned} XX &= X, \\ X + X &= 0, \\ X(Y + Z) &= XY + XZ. \end{aligned}$$

Note also that both $\times$ and $+$ are associative, commutative.

On the other hand, any formula of Boolean algebras can be translated into a formula of Boolean ring in the following conversion:

$$\begin{aligned} X \wedge Y &= XY, \\ X \vee Y &= XY + X + Y, \\ \neg X &= 1 + X. \end{aligned}$$

Therefore, we can assume every Boolean constraint is represented in polynomial equations on the corresponding Boolean ring.

If a part of a Boolean expression match the left hands of the above, the part can be replaced to the right hands. A Boolean expression α is said to be reduced to another expression β if α is converted to β by a finite number of applications of such replacement. For example $x^2y + x(y+z)$ can be reduced into xz. A Boolean expression called irreducible if such replacement cannot be applied to it. Any Boolean expression can be reduced to a unique irreducible form, which we call its normal form.

In this paper, we denote variables for elements of $\mathbf{B}$ by $x, y, z, \ldots$, Boolean polynomials by $A, B, C, \ldots$

We introduce some notations to present the new algorithm. Let $\prec$ be an arbitrary total order over variables.

Definition 2.1 We define that a variable y is a $\prec_x$-*variable* if y is smaller than x with respect to $\prec$. We define that a polynomial A is a $\prec_x$-*polynomial* if every variable in A is a $\prec_x$-variable.

Definition 2.2 Let $Ax \oplus B$ denote a Boolean polynomial $Ax + B$ and also mean that both A and B are $\prec_x$-polynomials.

In what follows, we denote semantical equality by $=$ and syntactical equality by $\equiv$.

Definition 2.3 We define the *normal* Boolean polynomials recursively as follows.

1. Two constants 0,1 are *normal*.
2. If A and B are normal $\prec_x$-polynomials, then $Ax \oplus B$ also is *normal*.

Let *norm* denote the normalizing operation.

Example 2.4 If $c \prec b \prec a$,

$$\text{norm}((a \wedge b) + ((b \vee c) - c)) \equiv ba \oplus ((c+1)b)$$

Definition 2.5 We define a relation $\leq$ on Boolean polynomials in the lattice sense, that is,

$$A \leq B \Longleftrightarrow AB = A$$

Any Boolean equation $P = Q$ can be translated norm$(P+Q) \equiv Ax \oplus B = 0$. It is known that $Ax \oplus B = 0$ is satisfied if and only if $B \leq x \leq A + B + 1$ are satisfied. Therefore, if $AB = B$, which is equivalent to $B \leq A + B + 1$, is satisfied, we can satisfy $Ax \oplus B = 0$ by putting the value of x between A and $A+B+1$. Thus, we can decide the value of each variable from the least to the greatest according to the order $\prec$ one by one. Boolean unification algorithm uses this property, in which variable x is substituted with value $(A+1)u + B$ where u is an extra variable. The substitution is an representation of $B \prec x \prec A + B + 1$.

3. New Boolean Constraint Solver

Our algorithm, which is based on the following idea, solves Boolean constraints incrementally. Let $Ax \oplus B = 0$ be a Boolean constraint which has been solved. This constraint means $B \leq x \leq A + B + 1$. Assume that another constraint $Cx \oplus D = 0$ is given. First, note that $AD + BC = 0$ since $C(Ax + B) = ACx + BC = 0$ and $A(Cx + D) = ACx + AD = 0$. Since the new constraints means $D \leq x \leq C + D + 1$, we get $B \vee D \leq x \leq (A+B+1) \wedge (C+D+1) = (BD + B + D) + (AC + A + C) + 1$. That is, a new constraint narrows the range of the variable x. From $BD + B + D \leq AC + A + C$ we denote the constraint for x by $(AC + A + C)x \oplus (BD + B + D) = 0$ (that is, $(A \vee C)x \oplus (B \vee D) = 0$). When $AC + A + C = 1$, the value of variable x is fixed on $BD + B + D$ by those of $\prec_x$-variables. The Boolean constraint $Cx \oplus D = 0$ includes Boolean constraints for $\prec_x$-variables, for example $(C+1)D = 0$. The included Boolean constraints also narrows the ranges of $\prec_x$ variables as well as x.

Now we present our algorithm more precisely.

Algorithm

When the algorithm is given an input set of Boolean constraints, **C**, it outputs a set of calculated Boolean constraints, **S**, if **C** is satisfiable.

In the algorithm, we keep the solved form of Boolean constraints in a set *DRule* or a set *NDRule*. *DRule* consists of Boolean constraints that take the form $x = A$ where A is a $\prec_x$-polynomial. The equation $x = A$ means that the value of variable x is determined by those of $\prec_x$-variables. If an equation $x = A$ is satisfied, we can substitute the polynomial A for the variable x in any formula except $x = A$. Hence let *DRule substitution* mean substituting each polynomial in the right hand side of an equation in *DRule* for the corresponding variable in the left hand side.

NDRule consists of Boolean constraints that take the form $Ax \oplus B = 0$. An equation $Ax \oplus B = 0$ means that the range of the variable x is $B \le x \le A + B + 1$.

We use *Temp* for keeping the formula being dealt with, and *Cons* for keeping the set of constraints still to be dealt with.

```
input C
$Cons \leftarrow \mathbf{C}$
$DRule, NDRule \leftarrow \emptyset$
while $Cons \neq \emptyset$
  choose an equation $F = G \in Cons$ and remove $F = G$ from $Cons$
  let $F'$ be the result obtained from $F + G$ by $DRule$ substitution
  $Temp \leftarrow \mathrm{norm}(F')$
  while $Temp \not\equiv 0$ and $Temp \not\equiv 1$
    if $Temp \equiv x \oplus D$
      then substitute $D$ for $x$ of $DRule$ and $NDRule$
        normalize the substituted formulas
        if $Ax \oplus B = 0 \in NDRule$
          then $NDRule \leftarrow NDRule - \{Ax \oplus B = 0\}$
            $Temp \leftarrow \mathrm{norm}(AD + B)$
          else $Temp \leftarrow 0$
      end-if
      $DRule \leftarrow DRule \cup \{x = D\}$
    else let $Temp \equiv Cx \oplus D$ where $C \not\equiv 1$
      if $Ax \oplus B = 0 \in NDRule$
        then $NDRule \leftarrow NDRule - \{Ax \oplus B = 0\}$
          if $\mathrm{norm}(AC + A + C) \equiv 1$
            then substitute $BD + B + D$ for $x$ of $DRule$ and $NDRule$
              normalize the substituted formulas
              $DRule \leftarrow DRule \cup \{x = \mathrm{norm}(BD + B + D)\}$
              $Temp \leftarrow \mathrm{norm}(AD + BC)$
            else $NDRule \leftarrow NDRule \cup \{\mathrm{norm}((AC + A + C)x \oplus (BD + B + D)) = 0\}$
              $Temp \leftarrow \mathrm{norm}(ACD + BC + CD + D)$
          end-if
        else $NDRule \leftarrow NDRule \cup \{Ax \oplus B = 0\}$
          $Temp \leftarrow \mathrm{norm}(AB + B)$
      end-if
    end-if
  end-while
  if $Temp \equiv 1$
    then stop and fail
  end-if
end-while
S $\leftarrow DRule \cup NDRule$
output S
stop
```

The termination of the algorithm is obvious. The soundness is proved by the next two Lemmas.

Lemma 3.1 Let $(A+1)B = 0$. Then

$$\begin{array}{lcl} Ax \oplus B = 0 & & (A \vee C)x + (B \vee D) = 0 \\ Cx \oplus D = 0 & \Longleftrightarrow & ACD + BC + CD + D = 0 \end{array}$$

Lemma 3.2 Let $(A+1)B = 0$. Then

$$ACD + BC + CD + D = 0 \Longrightarrow ((A \vee C) + 1)(B \vee D) = 0$$

In the algorithm, when a new constraint $Cx \oplus D = 0$ comes, we replace $Ax \oplus B = 0$ in $NDRule$ with $(A \vee C)x \oplus (B \vee D) = 0$ and obtains a constraint equation $ACD + BC + CD + D = 0$ in the case that $C \not\equiv 1$ and $AC + A + C \not\equiv 1$. This process is verified by Lemma 3.1. The other cases are special cases of this. Lemma 3.2 guarantees that the assumption of Lemma 3.1 is kept true for every $Ax + B = 0$ in $NDRule$, throughout the execution of the algorithm. Hence, we can conclude the soundness of the algorithm:

Theorem 3.3 **C** is equivalent to **S**.

The algorithm successively composes constraints that consist of fewer variables. The constraints that take the form $x \oplus B = 0$ influence other formulas by unifying x with B, while the constraints that take the form $Ax \oplus B = 0$ do not. Let us consider the following example:

Example 3.4 If $a \prec b \prec c$,

Inputs	$ab = 0$, $abc = c$
Outputs	$(1 + ab)c = 0$, $ab = 0$

Because the constraint $ab = 0$ does not influence $(1 + ab)c = 0$ in our algorithm, we cannot get $c = 0$.

To relieve this somewhat we introduce the following special reduction.

Reduction The form Fx is reduced by the formula $Ax \oplus B = 0$ as follows.

$$\begin{array}{ll} Fx \rightarrow x + BF + B & (F \not\equiv 1,\ \mathrm{norm}(AF + A + F) \equiv 1) \\ Fx \rightarrow (A+1)Fx + BF & (F \not\equiv 1,\ \mathrm{norm}(AF + A + F) \not\equiv 1,\ \mathrm{norm}(AF) \not\equiv 0) \end{array}$$

Then we can get a canonical form of **C** in the sense of [Sakai 90], if we reduce both $\prec_y$-polynomials C and D of every formula $Cy \oplus D = 0$ in S such that $x \prec y$, by each constraint $Ax \oplus B = 0$ in **S**.

Example 3.5 If $a \prec b \prec c \prec d$,

Inputs		$ab = 0,\ b \doteq d,\ abc = c$
Outputs	(not reduced)	$(1 + ab)c = 0,\ ab = 0,\ d = b$
	(reduced)	$c = 0,\ ab = 0,\ d = b$

4. Comparison

In this section we will compare our new algorithm with the modified Buchberger algorithm and Boolean unification in an example applying to a digital circuit, 5-Queens problem and a puzzle by Lewis Carroll [Colmerauer 90]. Each algorithm is implemented as a solver of CAL system in ESP (Extended Self-contained Prolog) [Chikayama 84] on PSI (Personal Sequential Inference machine) that was developed at ICOT [Taki 84].

We have to notice that when we unify a variable with the corresponding polynomial, we do not use the unification function of ESP language (similar to Prolog), but we rewrite the variable to the formula and reform the whole rewritten formula for a technical reason.

Since the canonical forms of Boolean constraints in each solver are different from one another, the outputs from each solver also are different when there are more than one solutions. In the tables of the following examples, Buch means the modified Boolean Buchberger algorithm, Unif means Boolean unification and New means our new algorithm.

A Digital Circuit

We will show the first example applying to a digital circuit that counts the number of 1s in five input signals to the circuit, and outputs it in the form of a three-bit binary number. In the program, each $\wedge$ operation, each $\vee$ operation, each $+$ operation and each $\neg$ operation corresponds to an *and* component, an *or* component, an *exclusive-or* component, and a *not* component in the circuit respectively. Each variable in the program corresponds to each input signal or output signal of the circuit. We executed it four times changing the input signals and the output signals of the circuit.

The execution times (*msec*) are as follows.

Input signal	Output signal	Buch	Unif	New
1	Variable	36	65	74
Variable	000	1332	737	288
Variable	001	1317	843	266
Variable	Variable	9975	797	508

We show the output from each solver in the third execution, that is, the input signals to the circuit are variables a, b, c, d, e, and the output signal from the circuit is 001. In the outputs from Boolean unification, a term $p(X)$ means an extra variable.

```
Buchberger Algorithm
    e = 1+d+b+a+c
    b*d = 0
    a*d = 0
    a*b = 0
    c*d = 0
    c*b = 0
    c*a = 0

New Algorithm
    e = 1+c+a+b+d
    (a+b+d)*c = 0
    (b+d)*a = 0
    d*b = 0

Boolean Unification
    e = 1+(1+(1+(1+(1+p(-34))*p(-33)+p(-34))*p(-32)+(1+p(-34))...
    c = (1+p(-34))*p(-30)
    a = (((1+p(-34))*p(-33)*p(-32)+(1+p(-34))*p(-33))*p(-31)...
    b = ((1+p(-34))*p(-32)*p(-31)+(1+p(-34))*p(-32))*p(-30)...
    d = (1+p(-34))*p(-31)*p(-30)+(1+p(-34))*p(-31)
```

5-Queens problem

In the program each variable corresponds to each square on the 5×5 chessboard. The value of the variables corresponding to the squares on which queens are put is 1, the value of the other variables is 0. We executed the program in three conditions: (1) all solutions, (2) solutions putting a queen on the top left square, (3) solutions not putting a queen on the top left square. We could not obtain the execution time in Boolean unification, because it took too much time in our implementation.

The execution times (*msec*) are as follows.

Condition	Buch	New
1	152915	2044640
2	4076	851
3	85587	769622

Puzzle by Lewis Carroll

We have eighteen sentences about sixteen propositions a, b, ...

$(f \wedge \neg c) \rightarrow g$ $\quad$ $(a \wedge o) \rightarrow b$ $\quad$ $(k \wedge h) \rightarrow n$

$(g \wedge \neg d) \rightarrow \neg f$ $\quad$ $(p \wedge h) \rightarrow j$ $\quad$ $(g \wedge d) \rightarrow h$

$(\neg i \wedge \neg p) \rightarrow \neg n$	$(c \wedge f) \rightarrow h$	$(n \wedge j) \rightarrow (g \wedge l)$
$(k \wedge g) \rightarrow i$	$(p \wedge c \wedge l) \rightarrow e$	$(k \wedge \neg a \wedge \neg b) \rightarrow \neg f$
$(n \wedge \neg c) \rightarrow \neg i$	$(a \wedge m \wedge d) \rightarrow g$	$(g \wedge j) \rightarrow \neg e$
$(n \wedge d) \rightarrow p$	$(\neg o \wedge \neg m) \rightarrow \neg k$	$(i \wedge h) \rightarrow (g \vee j)$

The problem is to find the relations among the designated propositions, if any. Let us try this problem by the modified Boolean Buchberger algorithm and our new algorithm. We represent each proposition by a variable, translate each sentence to a Boolean constraint, and let the order of the designated variables be lower than others. Then we can get some formulas that consists of only the designated variables, if any.

For example, among three variables a, f, i, there are no relation, while when we designate three variables f, n, p, we get $n * p * f = n * f$ by the modified Boolean Buchberger algorithm, and $(f * p + f) * n = 0$ by our new algorithm.

The execution times (*msec*) are as follows.

Propositions	Buch	New
a, f, i	334821	379
f, n, p	278303	291

5. Conclusion

We presented a new Boolean constraint solving algorithm that is based on the similar concept to Boolean unification. The algorithm has some advantages to other solvers. For example, it uses no extra variables, and computes a canonical form of Boolean constraints in the sense of [Sakai 90]. Moreover, as shown in the previous section, our new algorithm is fast in problems such as the first and the third examples, in which constraint relations are sparse. Our algorithm seems to be slower than the modified Boolean Buchberger algorithm for problems with dense constraint relations such as N-Queens.

Recently, we have been developing SetCAL, a solver of set constraint [Sato 90]. It can deal with finite or co-finite sets, and the symbols such as $\in$, $\subseteq$, $\{\cdot\}$, $\cap$, $\cup$, $\cdot^c$, and so on. The current solver is based on the modified Boolean Buchberger algorithm. However, if our new algorithm is used in it, it is expected to solve problems faster in many cases.

References

[Büttner 87] W. Büttner, H. Simonis : Embedding Boolean Expressions into Logic Programming, *J. Symbolic Computation* 4, pp.191–205, 1987.

[Chikayama 84] T Chikayama : Unique Features of ESP, *Proc. Fifth Generation Computer Systems 84*, pp.292–298, 1984.

[Colmerauer 90] A. Colmerauer : An Introduction to Prolog III, *CACM* 33, pp.69–90, 1990.

[Halmos 63] P.R. Halmos : *Lectures on Boolean Algebras*, D.Van Nostrand Company, 1963.

[Martin 86] U. Martin, T. Nipkow : Unification in Boolean Rings, *Proc. 8th Conf. on Automated Deduction*, *LNCS* 230, pp.506–513, 1986.

[Sakai 89] K. Sakai, A. Aiba : CAL: A theoretical background of constraint logic programming and its applications, *J. Symbolic Computation* 8, pp.589–603, 1989.

[Sakai 90] K. Sakai, Y. Sato : Application of Ideal theory to Boolean constraint solving, *Proc. Pacific Rim International Conference on Artificial Intelligence 90*, 1990.

[Sakai 91] K. Sakai, Y. Sato, S. Menju : *Boolean Gröbner base*, to appear (1991).

[Sato 91] Y. Sato, K. Sakai, S. Menju : *SetCAL - a solver of set constraint in CAL system*, to appear (1991).

[Taki 84] K. Taki, et al. : Hardware Design and Implementation of the Personal Sequential Machine (PSI), *Proc. Fifth Generation Computer Systems 84*, pp.398–409, 1984.