# 求解布尔与非线性数值约束相混合的约束问题*

季晓慧[1,2], 张 健[1+]

[1](中国科学院 软件研究所 计算机科学重点实验室,北京 100080)

[2](中国科学院研究生院,北京 100049)

## Solving Boolean Combinations of Nonlinear Numerical Constraints

JI Xiao-Hui[1,2], ZHANG Jian[1+]

[1](Laboratory of Computer Science, Institute of Software, The Chinese Academy of Sciences, Beijing 100080, China)

[2](Graduate School of the Chinese Academy of Sciences, Beijing 100049, China)

+ Corresponding author: Phn: +86-10-62644790, E-mail: zj@ios.ac.cn

Received 2003-08-01; Accepted 2004-03-01

**Ji XH, Zhang J. Solving Boolean combinations of nonlinear numerical constraints. *Journal of Software*, 2005,16(5):659-668.** DOI: 10.1360/jos160659

**Abstract**: Constraints involving Boolean and numerical variables are used widely, but it is difficult to solve especially when they contain nonlinear numerical expressions. Many existing methods for solving such constraints are incomplete. A new method is presented in this paper to solve Boolean combinations of nonlinear numerical constraints completely. This method transforms the nonlinear constraints into a special-formed optimization problem to solve them. A prototype tool is implemented, and some experiments are made. The experimental results show that the method is effective.

**Key words**: constraint solving; nonlinear numerical constraints; global optimization; interval methods

**摘 要**: 布尔与数值变量相混合的约束问题有着广泛的应用,但是当约束中的数值变量间存在非线性关系时该问题求解起来十分困难.目前的许多求解方法都是不完备的,即这些方法不能完全肯定某些包含非线性数值表达式的约束是否能够成立.针对这种问题,提出了将非线性数值约束转化为特殊形式的优化问题,采用全局优化算法对其进行求解的方法.已经实现了一个基于此方法的原型工具.实验结果表明,该方法能够有效地求解非线性混合约束问题,并且总能够得到该约束条件是否可满足的结果.

**关键词**: 约束求解;非线性数值约束;全局优化;区间方法

**中图法分类号**: TP301 **文献标识码**: A

* Supported by the National Science Fund for Distinguished Young Scholars under Grant No.60125207 (国家杰出青年科学基金)

**JI Xiao-Hui** was born in 1977. She is a Ph.D. candidate at Institute of Software, the Chinese Academy of Sciences. Her current research areas are constraint solving and optimization. **ZHANG Jian** was born in 1969. He is a professor at Institute of Software, the Chinese Academy of Sciences. His research areas are automated reasoning, constraint satisfaction, program testing and formal methods.

## 1 Introduction

Constraints involving both Boolean and numerical variables can be found in many real-world applications, ranging from formal verification of infinite state systems to planning with resources. In the analysis of state-based requirement specifications and in software test data generation, such constraints also need to be processed[1,2].

In general, determining the feasibility of such constraints is difficult, and it is even worse when the constraints contain nonlinear numerical expressions. In such cases, many existing constraint solvers delay the processing of nonlinear constraints, hoping that they will become linear after some variables are instantiated. However, if such instantiations do not occur, the user cannot get the correct answer[3].

In this paper, we present a method for solving Boolean combinations of nonlinear numerical constraints. It processes nonlinear constraints directly by transforming them into an optimization problem. We have implemented this method based on our previous tool BoNuS[1,2] and some experiments have been made. Preliminary experimental results show that our method is effective. Compared with other methods, our method can always give the user a definite answer.

The rest of this paper is organized as follows. In the next section, some concepts and notations are described, as well as the target problem. Then in Section 3, the details of our method are presented. In Section 4, an implemented tool is described, together with some examples. Finally, some related methods are compared with our approach, and conclusions are made.

## 2 Basic Concepts and Notations

To describe the problem clearly, we define some terms first. By *Boolean formula* (or *formula*) we mean a logical expression constructed from a set of Boolean variables using the logical operators AND ($\wedge$ i.e., conjunction), OR ($\vee$ i.e. disjunction), NOT ($\neg$ i.e., negation), IMP ($\rightarrow$ i.e., implication) etc. By *numerical constraint* we mean a relation (equality or inequality) between two mathematical expressions that may be linear or nonlinear. A *Boolean combination of nonlinear numerical constraints* can be viewed as an extension to a formula, in which a Boolean variable may be associated with a numerical constraint. By *domain* we mean a set in which a variable can take values from. For example, the domain for a Boolean variable is {FALSE, TRUE}.

In this paper, a constraint refers to a Boolean combination of nonlinear numerical constraints, unless specified differently. Our goal is to solve such constraints, i.e., to select a suitable value from each variable's domain so that each constraint holds. If we can find the values, we say that the constraints are *feasible*. Otherwise, they are *infeasible*.

For a simple example, let us look at the constraints:

$$\begin{cases}((x^2+y^2=2)\wedge b)\wedge((x^2+y<5)\vee(y<-2))\\ \neg(y<-2)\wedge e\rightarrow f\end{cases},$$

here $x, y$ are real variables, and $b, e, f$ are Boolean variables. The constraints are feasible if we can assign suitable real numbers to $x$ and $y$, and truth values to $b$, $e$ and $f$ such that each constraint holds. The solution may not be unique. It is obvious that $x=1, y=1, b=\text{TRUE}, e=f=\text{FALSE}$ is a solution to the constraints; and $x=0, y=-\sqrt{2}, b=e=f=\text{TRUE}$ is another solution. If the domain of $x$ is restricted to the set of real numbers that are greater than $\sqrt{2}$, then the above constraints are infeasible.

## 3 Algorithm for Solving the Constraints

It is quite challenging to solve the constraints defined in the previous section. Just as the name implies, Boolean combinations of numerical constraints have both the feature of the logic domain and that of the numerical domain. Thus neither using logic methods alone nor using mathematical methods alone can solve them. Moreover, the mathematical constraints can be linear or nonlinear and be equalities or inequalities. In addition, nonlinear problems are hard to solve. This paper extends our earlier work[1,2] which is incomplete when there are nonlinear numerical constraints.

### 3.1 Main algorithm

Our previous tool BoNuS[1,2] combines propositional logical reasoning with linear programming to solve the constraints. Its solving process can be abstracted as Fig.1.

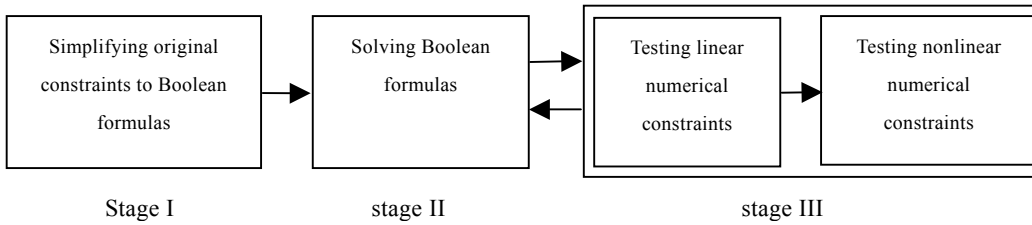


Fig.1 Main algorithm of BoNuS

In Stage I, it views numerical constraints as Boolean variables, and simplifies the original combined constraints to Boolean formulas. Then in Stage II, using a backtracking algorithm, it finds truth values for some Boolean variables. And in Stage III, the algorithm uses linear programming and bound propagation to solve the numerical constraints that have been assigned truth values, to ensure that the partial solution is not only valid in logic but also valid in the domain of real numbers.

In BoNuS, Stage III starts in two ways. First, it starts after all the Boolean variables in Stage II get their values. This means that all the numerical constraints are tested together. If the numerical constraints are determined to be feasible in Stage III, we get the result. But if they are infeasible, we have to go back to Stage II again and backtrack to get other truth values for numerical constraints and then go to Stage III again. The second way is that Stage III starts as soon as a numerical constraint (or the corresponding Boolean variable) gets a new value in Stage II. After the numerical constraints which got truth values are tested in Stage III, Stage II starts again to find a value for the next Boolean variable and then Stage III starts.

For example, let us study the constraint $a \wedge (x+y>2) \wedge \neg(x<1)$, in which $a$ is a Boolean variable, the domain of $x$ and $y$ is the whole set of real numbers. BoNuS first views $x+y>2$ and $x<1$ as Boolean variables, thus $a \wedge (x+y>2) \wedge \neg(x<1)$ is simplified to be $a \wedge b \wedge \neg c$. By using logical reasoning, we get the solution $a=b=\text{TRUE}, c=\text{FALSE}$. Then $\begin{cases} x+y>2 \\ x \geq 1 \end{cases}$ has to be checked. By using linear programming, we obtain $x=1$ and $y=2$. Then $a=TRUE, x=1, y=2$ is a solution to $a \wedge (x+y>2) \wedge \neg(x<1)$.

Linear programming can be used to solve linear constraints. To deal with nonlinear constraints, BoNuS combines linear programming with bound propagation. The latter is a useful technique, which tries to deduce new information from lower and upper bounds of variables. For example, if we know that the value of $x$ is between 3.0 and 5.2 and the value of $y$ is between 2.4 and 8.7, then we know that the constraint $2x-y>8$ does not hold, since the

value of the expression $2x-y$ should be within [−2.7,8]. Initially, the bounds of some variables may be $-\infty$ or $+\infty$. Although useful, bound propagation alone is inadequate for solving many nonlinear numerical constraints. And sometimes BoNuS can not give a correct answer.

The method presented in this paper is a complete constraint solving method. Its basic framework is the same as that in Fig.1, except that in Stage III, nonlinear numerical constraints are transformed into a special optimization problem and we use an interval-based method to solve it. In the following, we shall give the details of the method used in Stage III, namely how the numerical constraints are processed.

### 3.2 Solving numerical constraints

Generally the set of numerical constraints (denoted by *NC*) to be tested in Stage III can be divided into the set of linear numerical constraints (denoted by $NC_l$) and the set of nonlinear numerical constraints (denoted by $NC_{nl}$). If there are only linear constraints, we can easily decide whether they are feasible or not. But if nonlinear numerical constraints exist, it is more complicated.

Our algorithm for solving numerical constraints is like that in Fig.2.

1. Extract linear numerical constraints $NC_l$ from *NC*;
2. Solve $NC_l$ using linear programming;
3. If $NC_l$ are feasible, continue. Otherwise, the algorithm ends and the numerical constraints *NC* are infeasible;
4. Find each variable $x_i$ that occurs in nonlinear numerical constraints $NC_{nl}$ and get its bounds $[L_i,U_i]$ by using linear programming;
5. Transform nonlinear numerical constraints $NC_{nl}$ into constrained optimization problem P and use a global optimization method to solve it under the condition that each variable is in $[L_i,U_i]$;
6. If the optimal solution of P makes the value of the objective function to be 0, the numerical constrains *NC* are feasible and a solution is given. Otherwise, the numerical constraints *NC* are infeasible.

Fig.2 The algorithm for solving numerical constraint

In Fig.2, Steps 1~3 are easy to be implemented. In Step 4, to get the lower bound $L_i$ and the upper bound $U_i$ of $x_i$, we solve the following two linear programming problems[1]:

(1) Minimize $x_i$, subject to $NC_l$

(2) Maximize $x_i$, subject to $NC_l$

If the answer to problem (1) is unbounded, we assume $L_i$ is $-10^4$, and if the answer to problem (2) is unbounded, we assume $U_i$ is$10^4$. Then in step 5, we need only test nonlinear numerical constraints $NC_{nl}$ with each variable in $[L_i,U_i]$.

Generally speaking, it is more efficient to treat linear and nonlinear constraints separately, because linear programming[7] is quite mature and efficient for testing the feasibility of linear constraints, but there are few efficient methods for testing the feasibility of nonlinear constraints.

### 3.3 Solving nonlinear numerical constraints

Now we turn to Steps 5 and 6 for solving nonlinear numerical constraints. The nonlinear numerical constraints $NC_{nl}$ may contain *equality constraints* $gg_i(X)=0$ and *inequality constraints* $hh_i(X)\le 0$. To solve these constraints, in step 5 we transform them into a constrained optimization problem P.

An optimization problem looks like the following:

$$\begin{aligned} &\min && f(X)\\ &\text{s.t.} && g_i(X)=0,\ i=1,2,\ldots,p \end{aligned}$$

$$h_j(X) \le 0,\ j=1,2,\ldots,q \tag{1}$$

here $f(X)$ is called the *objective function*, $g_i(X)=0$ and $h_j(X)\le 0$ are called *constraints.* In particular, each $g_i(X)=0$ is called an *equality constraint* and each $h_j(X)\le 0$ is called an *inequality constraint.*

Our transformation is like the following. We let the objective function of the optimization problem be the sum of the square of each equality constraint's left-hand and let the constraints of the optimization problem be the inequality constraints. That is, $NC_{nl}$ is transformed into the following optimization problem:

$$\begin{aligned} \min \quad & f(X) = gg_1^{\,2}(X) + gg_2^{\,2}(X) + \ldots + gg_p^{\,2}(X) \\ \text{s.t.} \quad & hh_i(X) \le 0,\ i=1,2,\ldots,q \end{aligned} \tag{2}$$

It is easy to see that our transformed optimization problem P has inequality constraints only. And this is a great advantage, because for a global optimization method, if the optimization problem involves equality constraints, no matter how long the computation lasts it may fail to produce even one feasible result[8].

The criterion used in Step 6 for judging the feasibility of nonlinear numerical constraints is based on the following theorem.

**Theorem**. A sufficient and necessary condition for the feasibility of the numerical constraints $NC_{nl}$ is that we can find a point such that the value of $f$ in formula (2) is zero and $hh_i(X)\le 0$ holds.

*Proof.* Obvious. □

There are two special cases in this transformation, $p=0$ and $q=0$. If $p=0$, the transformed optimization problem is like the following:

$$\begin{aligned} \min \quad & f(X) = 0 \\ \text{s.t.} \quad & hh_i(X) \le 0,\ i=1,2,\ldots,q \end{aligned} \tag{3}$$

If $q=0$, the transformed optimization problem is like the following:

$$\min \quad f(X) = gg_1^{\,2}(X) + gg_2^{\,2}(X) + \ldots + gg_p^{\,2}(X) \tag{4}$$

It is an unconstrained optimization problem.

Generally, there are two classes of methods for solving optimization problem: *local methods* and *global methods*. A local method is essentially an iterated method and depends heavily on the current local information. So it sometimes cannot find the global minimum and even sometimes diverges[11]. Global methods[8] instead can find the global optimum by using the *interval* technology[9]. In an interval method, a real number $a$ is not expressed by the number itself but by an *interval* like $[\underline{a},\overline{a}]$, in which $\underline{a}\le a\le\overline{a}$ and we call $\underline{a}$ *lower bound* of $a$, $\overline{a}$ *upper bound* of $a$. The global optimization solving method is like that in Fig.3.

1. Input an initial interval vector $X$.
2. Decide whether the constraints in $X$ are feasible or not. If so, the algorithm continues. Otherwise, the algorithm ends and the optimization problem has no optimum.
3. In $X$, compute the current best rigorous upper bound $f^-$ of the objective function.
4. Bisect $X$ into $X_1$, $X_2$ according to some criterion.
5. Prune interval $X_i$ that cannot contain the optimum according to the objective function.
6. Return result(s) when each variable's interval is smaller than a predefined tolerance.

Fig.3 An interval-based global optimization algorithm

**Discussion**. There are two other methods for transforming the nonlinear constraints $\begin{cases} gg_i(x)=0, i=1,2,\ldots,p \\ hh_i(X)\le 0, i=1,2,\ldots,q \end{cases}$.

First, we can introduce slack variables to transform inequality constraints $hh_i(X)\le 0$ into equality constraints $hh_i(X)+y_i=0$, and then all equality constraints can be transformed into an unconstrained optimization problem like what we do above when $q=0$. But this method increases the number of variables, and in a global method this will decrease the efficiency of the algorithm[12].

Second, we can transform the equality constraints $gg_i(X)=0$ into the following two inequality constraints $\begin{cases} gg_i(x)-\varepsilon\le 0 \\ -gg_i(X)-\varepsilon\le 0 \end{cases}$, and then all inequality constraints can be transformed into a constrained optimization problem whose objective function is equal to 0 and which is like what we do above when $p$=0. But it does not make use of the information of the objective function, while our algorithm can use that information (see Step 5 in Fig.3). Moreover, although it transforms the equality constraints into inequality constraints, we can see that until the interval becomes $[-\varepsilon,\varepsilon]$, can the answer be found for the transformed equality constraints[8].

## 4 Implementation

### 4.1 Method for solving the transformed optimization problem

We have implemented our approach as an extension to BoNuS. The new tool is called BoNuS-2. Now we describe the method for processing the numerical constraints. Other implementation details can be found in Ref.[1]. Our tool calls the numerical toolkit Ctoolbox[10], which includes the global optimization method for the unconstrained optimization problem.

We use the simplex method presented in Ref.[7] to solve the linear constraints $NC_l$ and get the lower and upper bound for each variable which occurs in nonlinear constraints, as described in Section 3.2. For the optimization problem transformed from the nonlinear constraints $NC_{nl}$, we use the global method mentioned above to solve it. It is essentially the algorithm presented in Ref.[10] and its framework is like that in Fig.3 except for the absence of Step 2 because it is for unconstrained optimization problems. The algorithm is shown in Fig.4.

In Step 2 of the algorithm, the value of the objective function is computed with $X$ and the upper bound of the result interval $f^-$ is stored. $f^-$ is used in step 4 to prune the sub-interval $X_i$, in which the lower bound of the objective function is greater than $f^-$. It is easy to see that such a sub-interval cannot contain the minimum. It is obvious that the closer $f^-$ is to the minimum, the more sub-intervals can be pruned and the algorithm will be more efficient.

1. Input an initial interval vector $X$.
2. In $X$, compute the current best rigorous upper bound $f^-$ of the objective function.
3. Bisect the maximal interval of vector $X$ into $X_1$, $X_2$.
4. Prune interval $X_i$ that is impossible to contain optimum according to information of the objective function, using the interval Newton method.
5. Return result(s) when each variable's interval is smaller than a predefined tolerance.

Fig.4 The optimization algorithm used in Ctoolbox

Step 4 also uses the information of the first and second derivative of the objective function to prune the sub-intervals and accelerate the algorithm. Interval Newton method[8,10] is also used in Step 4.

As in Fig.3, Steps 2~4 are looped until all minima are found.

This global optimization method guarantees to find the global optimum[10]. Thus if we find the minimum is greater than 0 for the transformed optimization problem, we can say that the numerical constraints are infeasible. However, the algorithm is not completely appropriate for our purposes, and we make some modifications to it.

1. In Step 2, we specify the current best rigorous upper bound $f^-$ of the objective function to be $10^{-5}$ because the minimum of the objective function is known, i.e., zero ($10^{-6}$). Then $f^-$ is very close to the true minimum, so the efficiency of the algorithm can be enhanced greatly.
2. We add the constraint test. Our constraint test is based on the following rule:
   IF $Inf(hh_i(X)) > 0$ THEN this constraint cannot hold. Here $Inf(Y)$ denotes the lower bound of $Y$.
3. We change the stop criterion. Ctoolbox stops after finding all the minima, but this is not necessary for our problem. We stop after finding one point that makes the value of the objective function to be 0 and at the same time all constraints hold.

After these modifications, our algorithm for solving the transformed optimization problem is like that in Fig.5.

1. Input an interval vector $X$
2. Do constraints test in $X$
3. Set the current best rigorous upper bound $f^-$ of the objective function to be $10^{-5}$
4. Bisect the maximal interval of vector $X$ into $X_1$, $X_2$
5. Test the constraints in $X_i$
6. Prune interval $X_i$ that is impossible to contain optimum according to information of the objective function and interval Newton method
7. Get result(s) when the value of the objective function is 0 and each $Sup(hh_i(X))\leq 0$ holds

Fig.5 Our algorithm for solving the optimization problem

### 4.2 Some examples

The input of BoNuS-2 is a text file which specifies the constraints to be solved. In the input file, almost all the nonlinear expressions are allowed, including sin($x$), cos($x$), power($x$,$y$), ln($x$), exp($x$), sqrt($x$). They correspond to the mathematical functions $\sin(x), \cos(x), x^y, \ln(x), e^x, \sqrt{x}$ respectively. The output of BoNuS-2 is the answer that the constraints are feasible or not.

In the following examples, we give the input file, as well as the answer.

*Example* 1. Our first example is taken from Ref.[13], in which the problem is solved by combining symbolic and numeric methods. The input constraints and the result obtained by our tool are given below:

INPUT:

```
real x,y;
bool a = (power(x,2)–power(y,2)==0);
bool b = (x+y<25);
bool c = (x+y>0);
bool d = (x>= –100);
bool e = (x<=100);
{   and(a, b, c, d, e);   }
```

RESULT: Feasible ($x$=6.25, $y$=6.25)

*Example* 2. In this example, we have both Boolean and numerical variables:

INPUT:

```
real x,y;
```

bool *art_a* = (sin(*x*) > exp(*y*));
bool *art_b* = (power(*y*,2) > exp(*x*));
bool *a*;
bool *b*;
{ imp(or(and(*a*,*b*),*art_a*),*art_b*); }

RESULT: Feasible (*x*=0, *y*=0, *a* =FALSE, *b*=FALSE)

*Example* 3. This example is taken from Ref.[14]. The constraints are extracted from a path of a program to test if this path is feasible. In Ref.[14] the constraints are solved by using an entailment test, which is complex and needs more overhead. For BoNuS-2, the input and output are like the following:

INPUT:

int *x*,*y*,*z*,*t*1,*t*2,*u*;
bool *a* = (*z* == *x*∗*y*);
bool *b* = (*t*1 == 2∗*x*);
bool *c* = (*z* <= 8);
bool *d* = (*u* <= *x*);
bool *e* = (*t*2 ==*t*1−*y*);
bool *f* = (*t*2 <= 20);
{ and(*a*,*b*,*c*,*d*,*e*,*f*); }

RESULT: Feasible (*x*=0,*y*=0,*z*=0,*t*1=0,*t*2=0,*u*=0)

For each of the above examples, it takes our tool a fraction of a second to test the feasibility of the constraints, on a Pentium III with 600 MHz CPU.

We have tested our tool on other problems, such as test data generation and analysis of state-based requirement specifications. Due to the length limit of this paper, we shall not describe them here. More examples will be available on the website: http://lcs.ios.ac.cn/~zj/bonus2.html

In terms of computational complexity, nonlinear programming in general is considered intractable, and many of its subclasses are NP-hard[11]. The Boolean satisfiability problem is also NP-hard. Thus our problems are also intractable unless P=NP. However, in practice, the problem can be solved within a reasonable amount of time, if there are not too many variables. According to our experience, our tool can easily solve nonlinear constraints which have no more than ten real variables.

## 5 Related Work

Constraint logic programming (CLP) is a well-known paradigm for solving constraints, and many systems based on it have been developed, such as Prolog III and CLP($\Re$). However, all of them delay the nonlinear constraints hoping that the nonlinear expressions can become linear after some variables are instantiated. As we mentioned earlier, if such instantiations do not occur, the user cannot get the answer[3]. SICStus Prolog[4] is a more recent CLP system which has made some improvements. Yet it still can not solve some simple constraints like sin(*x*)=cos(*x*). Neither can it solve the constraint sin(cos(*x*))<0.5. In contrast, BoNuS-2 can solve such constraints easily.

QUAD-CLP($\Re$) is another CLP system, but just as its name implies, it can only deal with quadratic constraints[1]. Even for such constraints, it does not always give the user a definite answer. For example, given the following constraints:

(*x*∗*y* <=2); (2∗*x*>= *y*+3); (*y*>=1.2)

The answer of QUAD-CLP($\Re$) is “maybe”. This constraint system is easily solved by BoNuS-2, and the answer is

"infeasible".

The CLP systems mentioned above cannot *always* tell users whether constraints are feasible or not when there are nonlinear numerical constraints. But in such applications as test data generation, data dependence analysis etc., constraints that contain both Boolean variables and nonlinear numerical constraints may arise. So it is very necessary to find a method to solve Boolean combinations of nonlinear numerical constraints completely, namely a method that can *always* tell the user whether the given constraints are feasible or not. In particular, we may conclude that a statement in a program is not reachable if certain constraints are infeasible.

RISC-CLP (Real) is an outstanding system for solving nonlinear constraints. It can tackle those problems in which the relations of the numerical constraints can be other logic connectives besides AND. It combines constraint logic programming with two algebraic methods, *Partial Cylindrical Algebraic Decomposition* and *GrÖbner basis*[3], but it is inefficient and can only solve polynomial constraint problems. Moreover, the high complexity of its algorithms limits its usefulness[13].

Some systems combine several different solvers to solve the constraints[13,15]. For example in Ref.[13], it cooperates Prolog III, Maple V and Interlog together and each solver solves one kind of constraints. The efficiency of such a system depends on that of each solver and the task distribution among the solvers makes the whole system quite complex.

The new feature of our method is its ability to solve nonlinear constraints. In this respect, the most famous system we think is Numerica[11], which is an effective tool for solving constrained optimization problem. The implementation of Numerica uses a combination of numerical analysis and artificial intelligence. It is shown to outperform state of the art tools in many applications[11]. But Numerica can only process numerical constraints. Constraints having both Boolean and numerical variables are not included, although they can be integrated easily[11]. The examples given in Ref.[11] are all systems of equations, without inequalities. The focus of Numerica lies in optimization. Since Numerica is not available to us, we are not able to compare it with our tool.

A Newton type algorithm is presented in Ref.[5] to solve nonlinear numerical constraints. Although this kind of method is computationally efficient, it cannot provide guarantees for convergence[6]. By introducing slack variables, Maranas and Floudas[6] transform the numerical constraints into a global optimization problem whose global minimum solutions with zero objective value correspond to the solutions of the original numerical constraints[6].

Namely they transform the original problem $\begin{cases} h_j(X)=0, j\in N_E \\ g_k(X)\le 0, k\in N_I \\ X^L \le X \le X^U \end{cases}$, (where $N_E$ is the set of equalities, and $N_I$ is the set of inequalities) into the following problem:

$$\begin{aligned} \min_{x,s\ge 0} \quad & s \\ \text{s.t.} \quad & h_j(X)-s\le 0, j\in N_E \\ & -h_j(X)-s\le 0, j\in N_E \\ & g_k(X)\le 0, k\in N_I \\ & X^L \le X \le X^U \end{aligned}$$

Then they solve the transformed problem by successive refinement of a convex relaxation of the feasible region and subsequent solution of a series of nonlinear convex optimization problems. Obviously, this method is the second transforming method mentioned at the end of Section 3.3. It needs to exploit the mathematical structure of the problem and does not make use of the efficient linear programming tool as our method does.

## 6 Conclusion

In this paper, a new method that can solve Boolean combinations of nonlinear numerical constraints is presented. Compared with other methods, our method is simpler and more complete. The method is implemented in the tool BoNuS-2, which greatly enhances its ability of processing nonlinear numerical constraints.

It is obvious that the efficiency of our method is dependent on the optimization solving method we use. We note that more and more methods for solving the optimization problems have been proposed in the past few decades. As more powerful methods are invented, we expect that the efficiency of our approach will be increased accordingly.

**References**:

[1] Zhang J, Specification analysis and test data generation by solving Boolean combination of numeric constraints. In: Tse TH, Chen TY, eds. Proc. of the First Asia-Pacific Conference on Quality Software. Los Alamitos: IEEE Computer Society, 2000. 267−274.

[2] Zhang J, Wang X. A constraint solver and its application to path feasibility analysis. International Journal of Software Engineering & Knowledge Engineering, 2001,11(2):139−156.

[3] Hong H. RISC-CLP(Real): Logic programming with non-linear constraints over reals. In: Benhamou F, Colmerauer A, eds. Constraint Logic Programming: Selected Research. Cambridge: The MIT Press, 1993. 133−159.

[4] Swedish Institute of Computer Science (SICS). SICStus Prolog User's Manual, Release 3.10.1, 2003.

[5] Daniel JW. Newton's method for nonlinear inequalities. Numerical Mathematics, 1973,21:381−387.

[6] Maranas CD, Floudas CA. Finding all solutions of nonlinearly constrained systems of equations. Journal of Global Optimization, 1995,7(2):143−182.

[7] Dantzig G. Linear Programming and Extensions. New Jersey: Princeton University Press, 1963. 94−111.

[8] Horst R, Paralos PM. Handbook of Global Optimization. Boston: Kluwer Academic Publishers, 1995. 751−824.

[9] Moore RE. Methods and Applications of Interval Analysis. Philadelphia: SIAM Publication, 1979.

[10] Hammer R., Hocks M, Kulisch U, Ratz D. C++ Toolbox for Verified Computing I. Heidelberg: Springer, 1995. 312−342.

[11] Hentenryck PV, Michel L, Deville Y. Numerica: A Modeling Language for Global Optimization. Cambridge: The MIT Press, 1997. 1−16.

[12] Kearfott RB. On proving existence of feasible points in equality constrained optimization problems. Mathematical Programming, 1998,83(1):89−100.

[13] Marti P, Rueher M. A distributed cooperating constraints solving system. International Journal on Artificial Intelligence Tools, 1995,4(1,2):93−113.

[14] Gotlieb A, Botella B, Rueher M. A CLP framework for computing structural test data. In: Proc. of the 1st Int'l Conf. on Computational Logic, Constraints Stream. Heidelberg: Springer, 2000. 399−413.

[15] Audemard G, Bertoli P, Cimatti A, Kornilowicz A, Sebastiani R. A SAT based approach for solving formulas over Boolean and linear mathematical propositions. CADE-18, 2002. 195−210.