Detection of Infeasible Paths using Presburger Arithmetic

Kuniaki Naoi Naohisa Takahashi

NTT Software Laboratories
Musashino-shi, Tokyo, 180 Japan

Abstract

Detecting infeasible paths (IFPs) allows accurate computation of various kinds of program slices, and accurate detection of semantic errors that may occur when two variants of a program are merged. We propose a method of efficiently determining the truth of a prenex normal form Presburger sentence (P-sentence) bounded only by existential quantifiers, which is suitable for detecting IFPs. In this method, a coefficients matrix is converted into a triangular matrix based on the method proposed by D. C. Cooper. If the rank of the matrix is lower than the degree of the matrix, the matrix is triangulated by using a method for solving one linear equation with three or more unknowns, so that the matrix can be back-substituted. This paper also reports that an implementation of our method shows a slower increase in computation time than the previous method and reduces computation time by up to 3,000,000 times.

1 Introduction

Finding a calculation path that is never executed for any input is called infeasible path (IFP) detection [10, 2, 3, 13, 7, 9]. It makes various kinds of program analyses more accurate by analyzing each path [15]. IFP detection therefore allows accurate computation of various kinds of program slices [17, 18], and accurate detection of semantic errors that may occur when two variants of a program are merged [14]. One of the most important problems in IFP detection is to determine the satisfiability of a formula, called a path condition (PC), which indicates whether a path is executable [2].

Several theorem provers have been proposed which directly determine PC satisfiability [10, 7, 9]. Although these provide accurate determinations when they terminate, the PC satisfiability is undeterminable in general. Therefore, we have focused on conservative approximation of PC satisfiability, in which a PC is converted into another formula whose satisfiability is always determinable. The conservative approximation is such that the converted formula is determined to be satisfiable if the PC is satisfiable. For practical use, it is important to develop a conservative approximation method that finds as many unsatisfiable formulas as possible and to reduce the computation time for the method. These considerations led us to use the determination procedure for a Presburger sentence (P-sentence) [13] in order to determine the PC satisfiability.

Presburger arithmetic [5] contains the symbols (,), $\wedge$, $\vee$, $\neg$, $\forall$, $\exists$, $=$, $<$, $+$, 0, and 1. A P-sentence is a formula in Presburger arithmetic that has no free variables, is determinable, and for which the procedure for determining is known. In order to determine the satisfiability of a PC, first, we convert the PC into a formula in Presburger arithmetic with a conservative approximation. Next, all variables in the formula are bounded by existential quantifiers, where the bounded formula is a P-sentence. Then, if and only if the P-sentence is FALSE, the PC is determined to be unsatisfiable.

However, the truth of any P-sentence of length n is determined within deterministic time $2^{2^{2^{pn}}}$ for some constant $p > 1$ [5]. Our preliminary experiments using a prototype system for determining the truth of the P-sentences have also shown an exponential increase in computation time, when an identical variable appears more often in the same atom in a P-sentence. In IFP detection, an identical variable can repeatedly appear in the same atom of a P-sentence since each coefficient of a variable in a PC can take an arbitrary value. Therefore, the amount of computation for determining a P-sentence increases exponentially. As a result, the PC satisfiability may not be determined for a realistic program within a practical time [16].

This paper proposes a method that efficiently determines the truth of a prenex normal form P-sentence bounded only by existential quantifiers in order to solve the above-mentioned problem. In this method, the P-sentence is first converted into a conjunction form of divisibility relations. Next, a matrix that denotes variables in the relations is converted into a triangular matrix by a method similar to Gaussian elimination. Here, a theorem in number theory is used to make elements in the matrix 0. If the rank of the matrix is lower than the degree of the matrix, it is triangulated using a method for solving one linear equation with three or more unknowns [6], so that the matrix can be back-substituted. Then, the value of each variable for determining the truth of the P-sentence is calculated by a back-substitution process, and this calculation is performed for all variables in order.

Although this method is based on a method suggested by D. C. Cooper [5], it is different from his method in that the linear equation is solved when the rank is lower than the degree. Cooper pointed out that triangulation by number theory can be applied to determine the truth of the P-sentence that we are

0730-3157/96 $5.00 © 1996 IEEE

considering. However, he provided no detailed description of the decision procedure, and discussed neither the amount of computation for the procedure nor the effects of the triangulation. We implemented our procedure on the basis of Cooper's method, and found that a P-sentence cannot be determined because the matrix cannot be back-substituted when the rank of the matrix is lower than the degree.

We propose a method to determine the truth of the P-sentence regardless of rank, where the matrix is triangulated by using the method for solving one linear equation with three or more unknowns, so that the matrix can be back-substituted. The computation time of our method does not increase as quickly as time of Cooper's method, even if some coefficients in the PC become larger. As a result, computation is reduced when some coefficients become larger. A trial implementation of our method had computation time reduced by up to 3,000,000 times.

2 Presburger arithmetic

There is a class of determinable formulas called Presburger sentences (or P-sentences) [5, 8] defined in Presburger arithmetic as a subclass of formulas in a first-order predicate calculus.

2.1 Definition

First we will define the language of Presburger arithmetic, $\mathcal{L}^+$. The symbols of $\mathcal{L}^+$ are (,), $\wedge$, $\vee$, $\neg$, $\forall$, $\exists$, $=$, $<$, $+$, $-$, 0, 1, x, y, z, $\cdots$. Here, x, y, z, $\cdots$ are called variables. An expression is a finite sequence of symbols of $\mathcal{L}^+$.

A term is defined as follows:

1. Variables, 0, and 1 are terms.
2. If t_1 and t_2 are terms, then so are (t_1+t_2) and $-t_1$.
3. These are the only terms.

An atomic formula or atom is an expression of one of the forms $(t_1<t_2)$ or $(t_1=t_2)$, where t_1 and t_2 are terms.

A formula is defined as follows:

1. An atom is a formula.
2. If A and B are formulas and x is a variable, then $\exists xA$, $\forall xA$, $(A\vee B)$, $(A\wedge B)$ and $\neg A$ are all formulas.
3. These are the only formulas.

A Presburger sentence (or P-sentence) is a formula that has no free variables.

Next, the standard interpretation $\mathcal{J}^+$ for $\mathcal{L}^+$ is defined as follows:

1. The domain from which the variables take their values is the set of integers Z.
2. $=$, $<$, $+$, $-$, 0, and 1 all take their usual interpretations.

Under $\mathcal{J}^+$, an arbitrary P-sentence can be determined to be either TRUE or FALSE.

For convenience, (1+1) will be written as 2, ((1+1)+1) will be written as 3, and, in general, $(1+\cdots+1)$ (1 repeated k times) will be written as k. The term $t_1+(-t_2)$ will be written as t_1-t_2, and $t+\cdots+t$ (t repeated k times) will be written as kt. The symbols $\leq$, $\geq$, $>$, $|$ will also be used. These are all definable in terms of $<$ and $+$ alone. Here, $|$ denotes divisibility by a constant. That is, $c|b$ denotes that b is exactly divisible by c, where b is a term but c must be an integer. Note that $1|b$ is interpreted as TRUE for an arbitrary term b.

2.2 Determination of a P-sentence

We will now explain the determination procedure for the truth of a P-sentence using quantifier elimination, as proposed by D. C. Cooper.

The procedure is quantifier elimination which, given a formula of $\mathcal{L}^+$ of the form $\exists xF(x)$ where $F(x)$ is quantifier free, returns a formula F' of $\mathcal{L}^+$ which is equivalent to $\exists xF(x)$ but which contains no quantifiers and no variable x. Refer to [4] for the elimination algorithm.

To determine the truth of any P-sentence A, replace any expression $\forall x$ with the equivalent expression $\neg\exists x\neg$. Next, apply quantifier elimination to the innermost quantified subformulas of A, replacing subformulas of the form $\exists xF(x)$ with equivalent subformulas without quantifiers. Continue until all quantifiers have been eliminated. The resulting formula contains no variables and can therefore be determined immediately to be TRUE or FALSE.

3 Detection of IFPs

Whether a path is an IFP can be determined by computing a PC for the path using symbolic execution and then by determining the satisfiability of this PC using an automatic theorem prover. When the PC is not satisfiable, the path is determined to be an IFP.

3.1 Computation of a PC

We have proposed a method for computing PCs by demand-driven execution [13] for a path dependence flow graph (PDFG) [17, 18], which is a directed graph representation for an imperative program. This is efficient because it computes only the requisite expressions for the PC and because it iteratively uses the partial computation results in computing PCs.

When a program involves procedure calls, the PC computation process may not be terminated due to recursion. We do not analyze inter-procedures but only intra-procedures in order to avoid the above-mentioned problem. As a result, function application forms remain as a term within the PC after the PC computation process. Additionally, when a program involves loops, the PC computation process may not also be terminated in a way similar to that for procedure calls. We have proposed a method in which a loop is converted into a procedure call [13]. Because the process always terminates with procedure calls, the process with loops terminates.

3.2 Satisfiability problem for the PC

The PC may involve function application forms, operators, and various data type variables that are not allowed in a Presburger arithmetic. Variables with data types other than integer can be easily converted into integer variables as shown in [7]. We also proposed a method in which the same terms with operators other than Presburger arithmetic are converted into unique integer variables [13], so that the converted expression becomes an expression in Presburger arithmetic. For example, a term $f(a)+b$ and another term $f(a)+c$ can be converted into $x+b$ and $x+c$, respectively, if the scopes of both variables a are the same, where $f(a)$ is a function application form.

Satisfiability for the converted expression can always be determined. All the variables in the expression are bounded by existential quantifiers. The bounded expression is a P-sentence. If and only if the P-sentence is FALSE, the expression is unsatisfiable. Therefore, satisfiability of the P-sentence can be determined by conservative approximation. Although many methods have been proposed [2, 3, 7, 9, 10] for determining satisfiability, either they are not so accurate or they do not work automatically.

3.3 Determination of a P-sentence in IFP detection

The P-sentences treated in IFP detection are a prenex normal form in which all variables are bounded by existential quantifiers. In the previous method, each variable must be substituted whenever a quantifier is eliminated. On the other hand, in Cooper's method for determining the P-sentence, all the substitution is delayed until after all the quantifiers have been eliminated [5] as follows. This is because $\bigvee_{j=1}^{\delta}$, which is created when an existential quantifier is eliminated, and other existential quantifiers are commutative.

Using Cooper's method, determining PC satisfiability becomes the problem of determining the truth of the formula

$$\bigvee_{i_1=1}^{\delta_1}\bigvee_{i_2=1}^{\delta_2}\cdots\bigvee_{i_n=1}^{\delta_n}\left[\bigwedge_{j=1}^{n}\lambda_j\left|\sum_{k=1}^{n}\mu_{jk}i_k+\nu_j\right.\right] \qquad (1)$$

where μ_{jk} and ν_j are integers and λ_j is a positive integer. That is, the problem becomes that of computing at least one set of values in the bound

$$\bigvee_{i_1=1}^{\delta_1}\bigvee_{i_2=1}^{\delta_2}\cdots\bigvee_{i_n=1}^{\delta_n} \qquad (2)$$

to satisfy a conjunction form of all atoms

$$\bigwedge_{j=1}^{n}\lambda_j\left|\sum_{k=1}^{n}\mu_{jk}i_k+\nu_j,\right. \qquad (3)$$

each of which denotes divisibility relations.

4 High-speed determination of a P-sentence

Since a P-sentence is constructed from a PC in IFP detection, it can become a complicated form when there is an increase in the number of atoms, types of variables, occurrences of an identical variable in an atom, and other factors. Our preliminary experiment using a prototype system for determining the truth of a P-sentence has shown that extremely long computation time is needed when the number of occurrences is large. This section describes a procedure that, in order to reduce the computation time for determining the truth of a P-sentence in the above case, efficiently determines the truth of expression (1); that is, it efficiently computes sets of values to satisfy a conjunction form of all atoms in expression (1).

4.1 Cooper's suggestion

Cooper suggested the following method to compute the sets of values to satisfy expression (1) in [4]. A matrix (μ_{jk}) which denotes coefficients of variables in expression (2), i.e. a coefficient matrix, is converted into a triangular matrix by a method similar to Gaussian elimination. Then, all the solutions for expression (1) are obtained by a back-substitution process. Here, the following two theorems in number theory are used for triangulating and back-substituting.

Theorem 1 *Let i be a variable in a set of integers Z. Let A be a set of terms without i. Let $GCD(x,y)$ be the greatest common divisor of two integers x and y.*

$$m_1|a_1i+b_1 \wedge m_2|a_2i+b_2$$
$$\Leftrightarrow m_1m_2|di+b_1p_1m_2+b_2p_2m_1 \wedge d|a_2b_1-a_1b_2,$$

where $m_j\in Z$, $a_j\in Z$, $b_j\in A$ for j=1, 2, $d=GCD(a_1m_2,a_2m_1)$, and $p_1a_1m_2+p_2a_2m_1=d$.

p_1 and p_2 are found as a by-product if Euclid's algorithm is used to find the GCD.

Theorem 1 can convert a conjunction form of two atoms having a variable i in common into a conjunction form of an atom involving i and an atom without i.

Theorem 2 *Let i be a variable in a set of integers Z.*
$m|ai+b$ has solutions for i if and only if $d|b$,
the solutions are $i=-p(b/d)+t(m/d)$
for all integers t,
where $m\in Z$, $a\in Z$, $b\in Z$, $d=GCD(a,m)$,
and $pa+qm=d$.

Theorem 2 indicates the condition that a divisibility relation with a variable has a solution, and, if the solution exists, it is a general solution for the variable.

A determination procedure using these theorems is as follows. A coefficient matrix is converted into a triangular matrix using Theorem 1. Solutions for a variable are computed using Theorem 2 and a known bound of the variable. Then, the sets of the solutions are obtained by a back-substitution process.

4.2 Triangulation and back-substitution

The triangulation method suggested by Cooper cannot be applied to a matrix whose rank is lower than its degree, which we call a lower rank matrix. This section expands Cooper's method so that it can, and describes in detail a procedure for finding the sets of values that satisfy expression (1). When Gaussian elimination is used to obtain numerical solutions of linear equations, if the coefficient matrix of the equations is a lower rank matrix, the solution is inconsistent or indeterminate. On the other hand, when the sets of values to satisfy the conjunction of divisibility relations are computed, a lower rank matrix cannot be back-substituted for Cooper's suggestion because at least one of the diagonal elements is 0. We introduce a procedure for triangulating the matrix, which uses the method for solving a linear equation with three or more unknowns [6], so that a lower rank matrix can be back-substituted.

4.2.1 Triangulation by using a Theorem in number theory

A triangulation procedure for the coefficient matrix is described. This procedure converts lower triangular elements into 0 in the same order as in Gaussian elimination. Theorem 1 is used to make the elements 0. Additionally, if an expression is reducible, it is reduced in order to make the computation efficient.

The triangulation procedure converts expression (3) into an equivalent expression whose coefficient matrix is triangulated

$$\bigwedge_{j=1}^{r} \lambda'_j \Big| \sum_{k=j}^{n} \mu'_{jk} i_k + \nu'_j. \tag{4}$$

where $\lambda'_j \in Z$, $\mu'_{jk} \in Z$, and $\nu'_j \in A$ for all j $(1 \leq j \leq n)$ and for all k $(1 \leq k \leq n)$. Note that an atom led by λ'_j involves $i_j, i_{j+1}, \cdots, i_n$. In this sense, the matrix is said to be triangulated. In addition, r is the rank of the coefficient matrix (μ'_{jk}), where $1 \leq r \leq n$. Note that all elements of the lth row are 0 for arbitrary l $(r < l \leq n)$.

A procedure for making the elements 0 by using Theorem 1 is described in [19], which is different from Gaussian elimination.

4.2.2 Triangulation for a lower rank matrix

The triangulated matrix cannot be back-substituted if and only if it is a lower rank matrix in which a diagonal element of the kth row and kth column a_{kk} is 0 and an element of the kth row and jth column a_{kj} is not 0 for arbitrary j, where $k < j$. The method described here converts a matrix triangulated by **4.2.1** that *cannot* be back-substituted into a triangulated matrix that *can* be back-substituted. In this method, the coefficient matrix is converted using Theorem 3 defined below, so that the converted matrix can be back-substituted.

Theorem 3 *Let i_r be variables in Z.*

$$m \Big| \sum_{k=1}^{n} a_k i_k + b$$
$$\Leftrightarrow m \Big| \sum_{k=1}^{n} a_k i_k + b \wedge d_{l-1} \Big| \sum_{k=l}^{n} a_k i_k + b$$
$$\wedge d_1 = GCD(m, a_1) \wedge \bigwedge_{k=2}^{n-1} d_k = GCD(d_{k-1}, a_k),$$

where $m \in Z$, $a_r \in Z$, $b \in Z$, $n \geq 2$, $1 \leq r \leq n$, $1 < l \leq n$.

Proof. Shown in the appendix.

Theorem 3 can convert an atom that involves variables $i_1, i_2, \cdots, i_n$ into a conjunction of this atom and an atom that involves variables $i_l, i_{l+1}, \cdots, i_n$, where l is an arbitrary number in the range of $1 < l \leq n$.

In this method, if a matrix cannot be back-substituted because a_{jj} is 0, then a_{jj} is converted into a value other than 0 by Theorem 3, while a_{ik} is unchanged (where $1 \leq i < j$ and $1 \leq k \leq n$). Additionally, other diagonal elements, each of which is not 0, would never be 0. Furthermore, an element of the jth row and kth column remains 0 for arbitrary k where $1 \leq k < j$. As shown above, a matrix that cannot be back-substituted can be converted into a matrix that can by using Theorem 3.

4.2.3 Back-substitution

When the process in **4.2.1** and **4.2.2** is applied, expression (3) becomes

$$\bigwedge_{j=1}^{m} \lambda'_j \Big| \sum_{k=j}^{m} \mu'_{jk} i_k + \nu'_j, \tag{5}$$

where the coefficient matrix is triangulated and m is the degree of the matrix. Note that the matrix (μ'_{jk}) can be back-substituted.

A method of computing sets of values to satisfy expression (5) within the bound shown in expression (2) is shown in [5].

5 Detection examples of an IFP

Two examples of detecting an IFP in a program are given below.

5.1 Example without lower rank matrix

Here, we consider an example of detecting IFPs in the sample program shown in Fig. 1, where the matrix in the P-sentence determination procedure is not a lower rank matrix.

This program contains 3 *if*-statements and 8 $(= 2^3)$ paths. For simplicity of discussion, we now concentrate on the path for which conditions in both the first and the second *if*-statement are determined to be TRUE and the condition in the third *if*-statment is to

```
main(x,y)
int x,y;
{
    int u,v,w;
    u = 2 * x + 3 * y;
    v = 3 * x + y;
    w = -1 * x + 7 * y;
    ...
    if (u < 300) then {...} else {...}
    if (v > 300) then {...} else {...}
    if (w < 202) then {...} else {...}
    ...
}
```

Figure 1: Sample program.

be FALSE. Whether the path is an IFP is determined as follows. Because the PC of the path is

$$2x+3y<300 \wedge 3x+y>300 \wedge 7y-x>201, \quad (6)$$

the truth of the P-sentence

$$\exists x \exists y[2x+3y<300 \wedge 3x+y>300 \wedge 7y-x>201] \quad (7)$$

is determined. Because the truth is determined to be FALSE as follows, the path is determined to be an IFP. The procedure shown in **4.1** converts expression (7) into

$$\bigvee_{i_Y=1}^{21} \bigvee_{i_X=1}^{4998} [i_X+34i_Y<3906 \wedge 1666 \mid i_X+34i_Y+142800 \wedge 2499 \mid -i_X-153i_Y-142800]. \quad (8)$$

Using Theorem 1, we can convert expression (8) into

$$\bigvee_{i_Y=1}^{21} \bigvee_{i_X=1}^{4998} [i_X+34i_Y<3906 \wedge 4998 \mid i_X-204i_Y+142800 \wedge 7 \mid i_Y], \quad (9)$$

in which the coefficient matrix is triangulated. By using Theorem 2, we found that $7|i_Y$ has solutions and these are $i_Y=7t_Y$, where t_Y is an arbitrary integer. Furthermore, $1\leq 7t_Y \leq 21$ holds because $1\leq i_Y \leq 21$ from expression (9). As a result, $1\leq t_Y \leq 3$ holds because i_Y is an integer. Expression (9) is converted into

$$\bigvee_{t_Y=1}^{3} \bigvee_{i_X=1}^{4998} [i_X+924t_Y<3906 \wedge 4998 \mid i_X-1428t_Y+142800] \quad (10)$$

by substituting $i_Y(=7t_Y)$ in expression (9) by a back-substitution process. Here,

$$\bigvee_{i_X=1}^{4998} [i_X+924t_Y<3906 \wedge 4998 \mid i_X-1428t_Y+142800]$$

is

$$\bigvee_{i_X=1}^{4998} [i_X<2982 \wedge 4998 \mid i_X+141372] \quad (11)$$

if $t_Y=1$. By using Theorem 2, we can show that $4998 \mid i_X+141372$ has solutions, which are $i_X=-141372+4998t_X$, where t_X is an arbitrary integer.

$1\leq -141372+4998t_X \leq 4998$ holds because $1\leq i_X \leq 4998$ holds from expression (11). $29\leq t_X \leq 29$, that is, $i_X=3570$ holds because i_X is an integer. Because $i_X<2982$ is FALSE for $i_X=3570$, expression (11) is FALSE.

When $i_Y=2$ or $i_Y=3$, the equations are similarly FALSE. Therefore, expression (8), which is equivalent to expression (7), is determined to be FALSE.

5.2 Example with lower rank matrix

Let's consider another example of detecting IFPs, where the matrix in the P-sentence determination procedure is a lower rank matrix. We consider the program in which the statements `w = -1 * x + 7 * y` and `if (w < 202) then {...} else {...}` in a Fig. 1 program are converted into `w = -5 * x + 3 * y` and `if (w > -400) then {...} else {...}`, respectively. For simplicity of discussion, we consider the path in which the conditions of all the *if*-statements are TRUE. Because the PC of the path is

$$2x+3y<300 \wedge 3x+y>300 \wedge 3y-5x>-400, \quad (12)$$

the truth of a P-sentence

$$\exists x \exists y[2x+3y<300 \wedge 3x+y>300 \wedge 3y-5x>-400] \quad (13)$$

is determined. Expression (13) can be converted into

$$\bigvee_{i_Y=1}^{3} \bigvee_{i_X=1}^{14} [i_X+4i_Y<200 \wedge 42 \mid i_X-12i_Y+1200 \wedge 1 \mid i_Y], \quad (14)$$

the same as **5.1**. By replacing $1|i_Y$ by TRUE in expression (14), we get the coefficient matrix as

$$\begin{pmatrix} 1 & -12 \\ 0 & 0 \end{pmatrix},$$

where the matrix cannot be back-substituted because it is a lower rank matrix. Because

$$42 \mid i_X-12i_Y+1200$$
$$\Leftrightarrow 42 \mid i_X-12i_Y+1200 \wedge 1 \mid -12i_Y+1200,$$

as shown by Theorem 3, the coefficient matrix is converted into

$$\begin{pmatrix} 1 & -12 \\ 0 & 1 \end{pmatrix},$$

which can be back-substituted. As a result, expression (13) is determined to be TRUE by achieving the back-substitution process in a similar way to **5.1**. Therefore, the path is determined not to be an IFP, but a feasible path candidate.

6 Discussion

Here, we discuss qualitatively computation time in our method, which determines the truth of a prenex normal form P-sentence bounded only by existential quantifiers. We also discuss the conditions under which our method is useful.

6.1 Computation reduction

The number of substitutions of integer values for variables is up to $\prod_{j=1}^{n}\delta_j$ for computing the sets of values that satisfy expression (1) when using Cooper's method for determining a P-sentence. The candidates for the values to satisfy expression (1) can be thinned out by triangulation. Roughly speaking, solutions are values at intervals of m/d according to Theorem 2. Therefore, the number of substitutions for a variable i_j is reduced from δ_j to δ_j/α_j where α_j is a positive integer. That is, the total count becomes $\prod_{j=1}^{n}\delta_j/\alpha_j$.

Specifically, in the example of **5.1**, the number of substitutions is 3×1 by our method. However, the number of substitutions is 21×4998 by the previous method, because it is necessary to substitute variables (i_Y, i_X) by 21×4998 sets of values in expression (8).

6.2 Efficiency of our method

The determination procedure for a prenex normal form P-sentence bounded only by existential quantifiers is widely applied to automated correctness proof for specifications in various design fields [8], such as hardware [11], communication protocols [21], and programs [22, 23]. In these applications, the number of identical variables in the same atom is often 1. Even if this is not so, the number is always small. Therefore, the number of each δ_j, where $1\leq j\leq n$, in expression (2) is quite small. Thus, the efficiency of our method is low.

However, in IFP detection the P-sentence is constructed from a PC. Each coefficient of a variable in the PC takes an arbitrary value. Thus, an identical variable may repeatedly appear in an atom in the P-sentence. Accordingly, the number of each δ_j, where $1\leq j\leq n$, in expression (2) is generally larger. Therefore, computation will be greatly reduced when the number of coefficients is larger.

6.3 Experimental results

We present here experimental results obtained using a prototype system that we are constructing to examine the efficiency of our method. We currently have prototype systems that compute using the previous determination method for a P-sentence and our method. We use a Sun SPARC Station 20 HS 21 (CPU clock: 125 MHz, memory: 96 MB). Refer to [19] for details of the system.

Determining expression (7) takes about 48 seconds by the previous method, whereas it takes about 4.1 milliseconds by our method. That is, computation time is reduced by about 9,000 times.

Next, determining the P-sentence

$$\exists x\exists y[2x+3y>300 \wedge 3x+y<300 \wedge x-5y>300]$$

takes about 8 seconds by the previous method to get the result FALSE, whereas it takes about 5 milliseconds by our method. That is, computation time is reduced by about 1,600 times.

In addition, to examine more generalized efficiency, a P-sentence

$$\exists x\exists y[2x+3y<300 \wedge 3x+y>300 \wedge by-ax>201]$$

was determined where a and b are parameters ($0\leq a\leq19$ and $1\leq b\leq9$). The results of experiments using the previous methods are shown in Fig. 2. In comparison, our method took between 4 and 7 milliseconds. The computation time does not increase much with our method compared to Cooper's method, even when the number of coefficients in the PC increases. Therefore, our method is more efficient than the previous method in such cases. We examined the efficiency of our method by computing the ratio of the computation time by the previous method to that by our method, when one of two variables was fixed. Although the ratio increased and decreased repeatedly, a line could be drawn which nearly passes through each relative maximum point in a graph with a logarithmic scale for ratio. This line indicates that the relative maximum value in ratio increases exponentially. Figures 3 and 4 are examples of this evidence. In addition, when $(a,b)=(15,9)$ holds, the maximum ratio rises to 3,000,000.

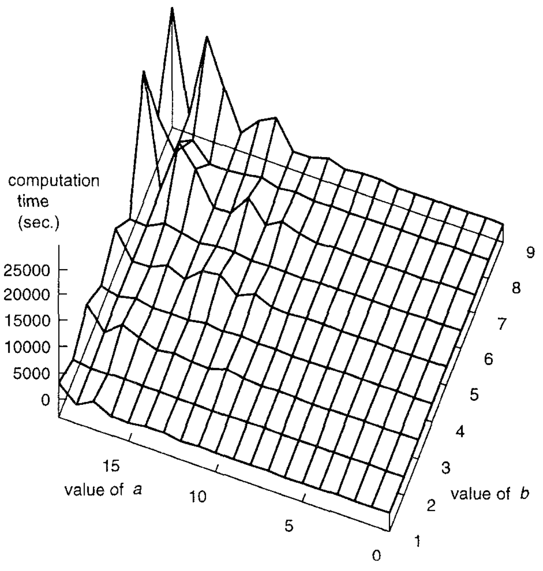

Figure 2: Computation time using previous method.

7 Related Work

Many researchers have tried to find ways to determine the truth of the kind of P-sentences that we have concentrated on, and they have treated it as a problem

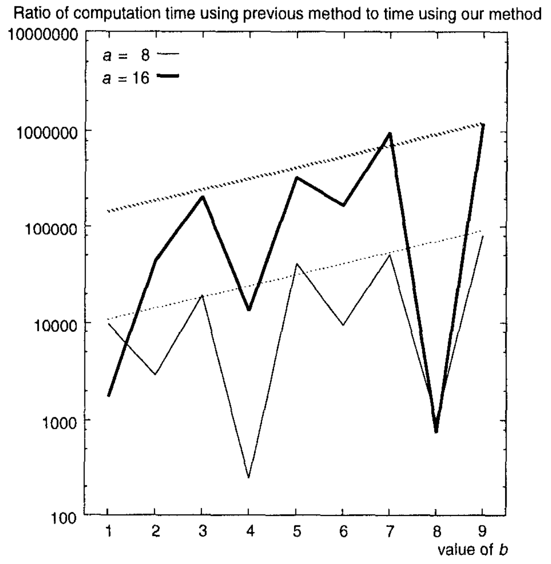

Figure 3: Ratio of computation time using previous method to time using our method (value of a is fixed).

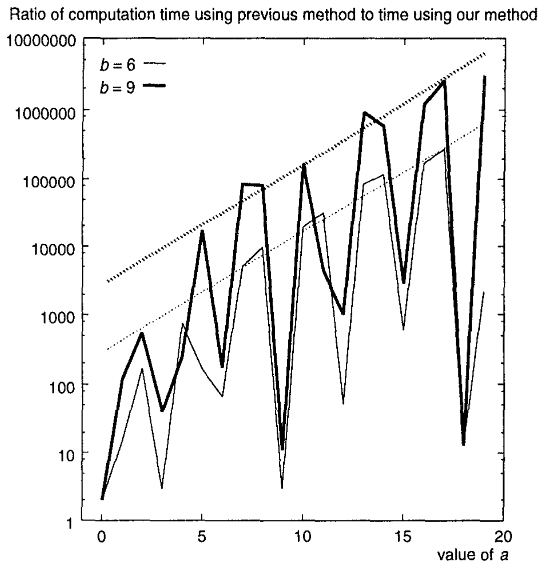

Figure 4: Ratio of computation time using previous method to time using our method (value of b is fixed).

of determining whether there exists a feasible solution to integer programming. Although integer programming is an NP-complete problem, Lenstra [12] showed that integer programming can be solved in polynomial time when the number of variables is fixed. Clarkson [4] proposed a faster algorithm than Lenstra's, when $n \gg d$ holds where n is the number of constraints and d is the number of variables. However, no program for the polynomial time algorithm is available.

The Omega test proposed by Pugh [24] is an efficient method in practice, for determining whether there exists a feasible solution to integer programming. This method is applied to array data dependence testing. An implementation of this method can be obtained via **ftp**; it determines the existence of a solution in less than 0.5 milliseconds on a 12-MIPS workstation when the number of both constraints and variables is small [24]. However, the computation time increases more rapidly as the numbers of constraints and variables increase, because the Omega test has exponential worst-case time complexity.

When IFPs are detected, the number of constraints corresponds to the number of *if* statements, and the number of variables corresponds to the number of variables appearing in each statement. Therefore, both numbers can be large even for a moderate size program. Consequently, it may take too long to determine the satisfiability for the Omega test. On the other hand, for our method, the increase in computation time is not so extreme, even though the computation time by our method is longer than that by the Omega test when the numbers of constraints and variables are small.

For example, the P-sentence

$$\begin{aligned}&\exists a \exists b \exists c \exists d \exists e \exists f\ [a+b+2c+2d+e+2f \geq 1\\ &\quad \wedge\ a-3b-2f \geq 1 \wedge\ -3a-b \geq 1 \wedge\ -a-c \geq 1\\ &\quad \wedge\ -b-3c+2e-2f \geq 1 \wedge\ -a-2e \geq 1\\ &\quad \wedge\ b-2d-f \geq 1 \wedge\ 4a+b-e \geq 1\\ &\quad \wedge\ a+c-e \geq 1 \wedge\ a \geq 1]\end{aligned} \tag{15}$$

is determined to be TRUE by our method in 95 milliseconds on a Sun SPARC Station 20 HS 21 (CPU: 125 MHz, Memory: 96 MByte), whereas it takes 200 milliseconds by the Omega test, which is approximately twice as long.

Another P-sentence

$$\begin{aligned}&\exists a \exists b \exists c \exists d \exists e \exists f\ [a+b+2c+2d+e+2f \geq 1\\ &\quad \wedge\ 2a-3b-2f \geq 1 \wedge\ -3a-2b \geq 1 \wedge\ -a-3c \geq 1\\ &\quad \wedge\ -b-3c+2e-2f \geq 1\\ &\quad \wedge\ -a-2e \geq 1 \wedge\ b-2d-3f \geq 1\\ &\quad \wedge\ 3a+b-e \geq 1]\end{aligned} \tag{16}$$

is determined to be FALSE by our method in 21 milliseconds, whereas it takes 2.4 seconds by the Omega test. In this case, our method is approximately 100 times faster than the Omega test.

8 Conclusion

We proposed a method of efficiently determining the truth of a prenex normal form P-sentence bounded only by existential quantifiers, which is used for detecting infeasible paths (IFPs).

In this method, the P-sentence is first converted into the conjunction form of divisibility relations. Next, a matrix that denotes coefficients of variables in the relations is converted into a triangular matrix by a method similar to Gaussian elimination. Here, a theorem in number theory is used to make elements in the matrix 0. If the rank of the triangulated matrix is lower than the degree of the matrix, the matrix is triangulated using a method for solving one linear equation with three or more unknowns, so that the matrix can be back-substituted. Then, values of each variable for the P-sentence to be satisfied are calculated in order by a back-substitution process. We also discussed the reduction of computation by our method, which was demonstrated by examples of determining and by experimental results using a prototype system.

Although the method we propose is based on a method suggested by D. C. Cooper, it is different from his method in that the linear equation is solved when the rank is lower than the degree. We have proved by implementing his suggestion that the P-sentence cannot be determined because the matrix cannot be back-substituted when the rank of the matrix is lower than the degree. We proposed a method to determine the P-sentence regardless of the rank, where the matrix is triangulated by using the method for solving one linear equation with three or more unknowns, so that the matrix can be back-substituted.

In addition, we have shown that our method is more effective when some coefficients are large, as in IFP detection, compared with when all coefficients are small, as in the correctness proof of communication protocol. Experimental results using a prototype system demonstrated the efficiency of our method.

We are conducting experiments with many examples to confirm the efficiency of our method. We will report the experimental results and discuss the amount of computation.

References

[1] Aho, A.V., Sethi, R., and Ullman, J.D.: "Compilers: Principles, Techniques, and Tools," Addison-Wesley (1986).

[2] Clarke, L.A.: "A system to generate test data and symbolically execute programs," IEEE Transactions on software engineering, Vol. SE-2, No. 3, pp. 215–222 (Sept. 1976).

[3] Clarke, L.A. and Richardson, D.J.: "Applications of symbolic evaluation," Journal of Systems and Software, Vol. 5, No. 1, pp. 15–35 (1985).

[4] Clarkson, K.L.: "Las Vegas algorithms for linear and integer programming when the dimension is small," Journal of the Association for Computing Machinery, Vol. 42, No. 2, pp. 488–499 (March 1995).

[5] Cooper, D.C.: "Theorem proving in arithmetic without multiplication," Machine Intelligence, No. 7, pp. 91–99 (1972).

[6] Dickson, L.E: "History of the theory of numbers," Vol. II, Carnegie Institution of Washington (1952).

[7] Goldberg, A., Wang, T.C., and Zimmerman, D.: "Application of feasible path analysis to program testing," Proceedings of the 1994 International Symposium on Software Testing and Analysis (ISSTA), pp. 80–94 (Aug. 1994).

[8] Higashino, T., Kitamichi, J., and Taniguchi, K.: "Presburger Arithmetic and its Application to Program Developments," Computer Software, Vol. 9, No. 6, pp. 31–39 (Jun. 1992) (in Japanese).

[9] Jasper, R., Brennan, M., Williamson, K., Currier, B., and Zimmerman, D.: "Test data generation and feasible path analysis," Proceedings of the 1994 International Symposium on Software Testing and Analysis (ISSTA), pp. 95–107 (Aug. 1994).

[10] King, J.C.: "Symbolic execution and program testing," Communications of the ACM, Vol. 19, No. 7, pp. 385–394 (July 1976).

[11] Kitamichi, J., Morioka, S., Higashino, T., and Taniguchi, K.: "Automatic correctness proof of the implementation of synchronous sequential circuits using an algebraic approach," Proceedings of the 2nd International Conference on Theorem Provers in Circuit Design (TPCD '94) (Kropf, T. and Kumar, R. Eds.), Vol. 901 of Lecture Notes in Computer Science, pp. 165–184, Springer Verlag (1995).

[12] Lenstra, H.W. Jr.: "Integer programming with a fixed number of variables," Mathematics of Operations Research, Vol. 8, No. 4, pp. 538–548 (Nov. 1983).

[13] Naoi, K. and Takahashi, N.: "Detection of Unfeasible Paths with a Path Dependence Flow Graph," Systems and Computers in Japan, Vol. 25, No. 10, pp. 1–14 (1994).

[14] Naoi, K. and Takahashi, N.: "Program Analysis with a Path Dependence Flow Graph," NTT R&D, Vol. 42, No. 8, pp. 1007–1016 (Aug. 1993) (in Japanese).

[15] Naoi, K. and Takahashi, N.: "A program Analysis System with a Path Dependence Flow Graph," Proceedings of the 47th Annual Convention IPS Japan, 5D–9 (Nov. 1993) (in Japanese).

[16] Naoi, K. and Takahashi, N.: "Computation reduction in infeasible path detection that uses a path dependence flow graph," Proceedings of the 48th Annual Convention IPS Japan, 6G–2 (Mar. 1994) (in Japanese).

[17] Naoi, K. and Takahashi, N.: "Program slicing using a path dependence flow graph," The transactions of the institute of electronic, information and communication engineers D-I, Vol. J78–D–I, No. 7, pp. 607–621 (July 1995) (in Japanese).

[18] Naoi, K. and Takahashi, N.: "Program slicing using a path dependence flow graph," Systems and Computers in Japan, (1996) (to appear).

[19] Naoi, K. and Takahashi, N.: "Technique for reducing computation for detecting infeasible paths using Presburger arithmetic," Technical report of the institute of electronic, information and communication engineers, SS95-19, (July 1995) (in Japanese).

[20] Oppen, D.C.: "A $2^{2^{2^{pn}}}$ upper bound on the complexity of Presburger arithmetic," Journal of computer and system sciences, No. 16, pp. 322–332 (1978).

[21] Li, X.D., Higashino, T., and Taniguchi, K.: "An automatic Derivation of Test Cases for LOTOS Expressions with Data Parameters," The transactions of the institute of electronic, information and communication engineers B-I, Vol. J75-B-I, No. 11, pp. 734–743 (Nov. 1992) (in Japanese).

[22] Morioka, S., Okano, K., Higashino, T., and Taniguchi, K.: "Semi-automatic proof of correctness of refinements of abstract state machine style description using decision procedure for Presburger sentences," 11th Conference Proceedings, Japan Society for Software Science and Technology, E7-4, pp. 361-364 (Oct. 1994) (in Japanese).

[23] Morioka, S., Okano, K., Higashino, T., and Taniguchi, K.: "Hierarchical Design of Stock Management Program using Relational Algebra and Its Correctness Proof," Transactions of Information Processing Society of Japan, Vol. 36, No. 5, pp. 1091-1103 (May 1995) (in Japanese).

[24] Pugh, W.: "A practical algorithm for exact array dependence analysis," Communications of the ACM, Vol. 35, No. 8, pp. 102–114 (Aug. 1992).

Appendix

Proof of Theorem 3

$m \mid \sum_{k=1}^{n} a_k i_k + b$

$\Leftrightarrow -mh + \sum_{k=1}^{n} a_k i_k + b = 0,$

where h is a variable in Z.

$\Leftrightarrow -mh + \sum_{k=1}^{n} a_k i_k + b = 0$
$\wedge -mh + a_1 i_1 = d_1 j_1$
$\wedge \bigwedge_{k=2}^{n-1} d_{k-1} j_{k-1} + a_k i_k = d_k j_k$
$\wedge d_{n-1} j_{n-1} + a_n i_n = -b$
$\wedge d_1 = GCD(m, a_1)$
$\wedge \bigwedge_{k=2}^{n-1} d_k = GCD(d_{k-1}, a_k),$

where j_k is a variable in Z.

(This division is known as a method for solving one linear equation with three or more unknowns [6].)

$\Leftrightarrow -mh + \sum_{k=1}^{n} a_k i_k + b = 0$
$\wedge d_{l-1} j_{l-1} + \sum_{k=l}^{n} a_k i_k + b = 0$
$\wedge d_1 = GCD(m, a_1)$
$\wedge \bigwedge_{k=2}^{n-1} d_k = GCD(d_{k-1}, a_k),$ where $1 < l \leq n$.

(This is because j_k is eliminated from $d_{k-1} j_{k-1} + a_k i_k = d_k j_k$ and $d_{n-1} j_{n-1} + a_n i_n = -b$, where $l \leq k < n$.)

$\Leftrightarrow m \mid \sum_{k=1}^{n} a_k i_k + b$
$\wedge d_{l-1} \mid \sum_{k=l}^{n} a_k i_k + b$
$\wedge d_1 = GCD(m, a_1)$
$\wedge \bigwedge_{k=2}^{n-1} d_k = GCD(d_{k-1}, a_k)$ □