Systems and Computers in Japan, Vol. 30, No. 9, 1999
Translated from Denshi Joho Tsushin Gakkai Ronbunshi, Vol. J80-D-I, No. 3, March 1997, pp. 269–281

Detection of Infeasible Paths Using Presburger Arithmetic

Kuniaki Naoi and Naohisa Takahashi

NTT Software Laboratories, Musashino, Japan 180

SUMMARY

An efficient method is proposed to determine the truth of a prenex-normal form Presburger sentence bounded only by existential quantifiers (EPP-sentence), which is used for detecting infeasible paths (IFPs). Detection of IFPs makes it generally possible to perform various kinds of program analyses more accurately along a computation path. In conventional determination methods for the truth of general Presburger sentences, there are cases where the amount of computation is extremely large and IFPs cannot be detected within a practical time. In the proposed method, the matrix of coefficients for variables (coefficient matrix) is triangulated using a theorem in number theory. If the rank of the triangulated coefficient matrix is less than the degree of the matrix, the coefficient matrix is triangulated using a method for solving one linear equation with three or more unknowns. Furthermore, the truth of the EPP-sentence is determined using back-substitution. The proposed method is particularly effective when the absolute values of the coefficients associated with the variables in the EPP-sentence are large; that is, when the absolute values of coefficients for path condition in IFP detection are large. We confirm that an implementation of the proposed method reduces computation time by a factor of up to 3 million times compared with the previous method. ©1999 Scripta Technica, Syst Comp Jpn, 30(9): 74–87, 1999

Key words: Infeasible path; Presburger arithmetic; program analysis; satisfiability problem; theorem prover.

1. Introduction

A computation path never executed in a program is called an infeasible path (IFP) [1–5]. Detection of IFPs makes it possible to perform various kinds of program analyses more accurately [8, 9] along computation paths, such as accurate computation of program slices [6] and accurate detection of semantic errors that may occur when two variants of a program are merged [7]. Whether a path is an IFP or not is determined by whether the input variables satisfy are required path conditions in order to execute a program along a path. A path that has unsatisfiable path conditions is an IFP [3].

The satisfiability problem of path conditions is undeterminable in general. Therefore, there are path conditions whose satisfiability cannot be determined [10]. Thus, it is impossible to determine whether some paths are IFPs or not. However, it is sufficient to find as many IFPs as possible in the program in order to make accurate analyses along paths. That is, it is sufficient to determine the unsatisfiability of path conditions in as many cases as possible.

For that purpose, we first select as broad a class as possible whose satisfiability can be determined, and we precisely determine the satisfiability of path conditions for that class. Next, for path conditions that do not belong to the class, we determine their satisfiability by conservative approximation [3]. The conservative approximation makes a judgement of unsatisfiability only when unsatisfiability of path conditions can be guaranteed. Furthermore, this approximation is made so that as many cases as possible can be determined to be unsatisfiable.

We previously proposed a class of formulas on Presburger arithmetic (PA) [11, 12] whose satisfiability can be determined [3]. The satisfiability of such formulas can be

CCC0882-1666/99/090074-14

determined using determination method of the truth of EPP-sentences. An EPP-sentence is a formula defined on the PA. Additionally, we proposed a conservative approximation method that transforms path conditions into formulas on the PA and determines the satisfiability of the transformed formulas [3].

If the number of variables is not fixed, the problem of determining the truth of EPP-sentences is NP-complete [13, 14]. Additionally, there has been no demonstration of whether the determination of the truth of the EPP-sentence is NP-complete [13] if the number of variables is fixed to an arbitrary number. If the number of variables is fixed, there is a possibility that the truth of EPP-sentences can be determined within a deterministic polynomial time. However, our experiment with a prototype system showed that, even if the number of variables is fixed, computation time increases in exponential order if EPP-sentences become slightly more complex, e.g., when absolute values of the coefficients of the variables in an EPP-sentence become large [10]. In particular, in IFP detection, an EPP-sentence is generated from path conditions, and the absolute values of the coefficients of the EPP-sentence variables tend to become large, because the coefficients of the variables in the path conditions can take various values. Thus, there are cases where IFPs cannot be detected within a practical time for real programs by existing determination method [15].

In this article, we propose a method for high-speed determination of the truth of EPP-sentences that is suitable for IFP detection in order to solve the above-mentioned problem. In the proposed method, we first transform variables bounded by existential quantifiers to variables with restricted scope of values by quantifier elimination. Next, we triangulate the coefficient matrix denoting the transformed variables by a method similar to Gaussian elimination. We use a theorem in number theory to make the matrix elements zero. Furthermore, when the rank of the triangulated matrix is less than the degree of the matrix, we triangulate the coefficient matrix by a method for solving one linear equation with three or more unknowns [16]. Additionally, we restrict the scope of variables using back-substitution. Finally, we determine the truth of the EPP-sentence by using a combination of values for restricted bounds of the variables.

The proposed method depends on that suggested by Cooper [1], but it differs from Cooper's method in that we use a method for solving one linear equation with three or more unknowns. Cooper points out that the computation time for detecting the truth of the EPP-sentences we are considering can be reduced by triangulation of the coefficient matrix using a theorem in number theory. However, he provides no detailed description of the decision procedure and no concrete effect of triangulation. We implement a triangulation algorithm as described by Cooper, and we show that there are cases where back-substitution cannot be applied and the truth of EPP-sentences cannot be determined when the rank of the triangulated coefficient matrix is less than the degree of the matrix. In the proposed method, we transform the coefficient matrix to which back-substitution cannot be applied into one to which it can be applied, using a method for solving one linear equation with three or more unknowns. Thus, it becomes possible to determine the truth of EPP-sentences, regardless of the rank of the coefficient matrix.

The computation time of the proposed method increases less rapidly even if the absolute values of the coefficients in the EPP-sentence are large. That is, the proposed method greatly reduces the computation time needed for determining the truth of EPP-sentences compared with the previous method when the absolute values of the coefficients are large. Additionally, we confirm that an implementation of the proposed method reduces computation time by a factor of up to 3 million compared with the previous method.

This article is constructed as follows. We define Presburger arithmetic and EPP-sentences in Section 2. In Section 3, we describe a method of detecting IFP, and in Section 4, we describe a high-speed method for determining the truth of EPP-sentences. In Section 5, we present examples of IFP detection. In Section 6, we discuss the proposed method, and, lastly, we survey related studies in Section 7.

2. Definition of PA and EPP-Sentences

There is a class of determinable formulas called Presburger sentences (or P-sentences) defined in Presburger arithmetic (PA) as a subclass of formulas in a first-order predicate calculus [11, 12].

2.1. Definition of PA

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

A term is defined as follows:

(1) variables, 0, and 1 are terms.

(2) If t_1 and t_2 are terms, then $(t_1 + t_2)$ and $(-t_1)$ are terms.

(3) Only sequences produced applying (1) and (2) are 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 $(A \wedge B)$, $(A \vee B)$, and $\neg A$ are all formulas.

(3) If A is a formula and x is a variable, then $\exists xA$ and $\forall xA$ are formulas.

(4) Only sequences produced by applying (1) through (3) are formulas.

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

Next, the standard interpretation T^+ for L^+ is defined as follows:

(1) The domain from which the variables take their values is the set of integers Z.

(2) The symbols =, <, +, −, 0, and 1 all take their usual interpretations.

Under T^+, 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 + . . . +1) (where 1 is repeated k times) will be written as k. The term $t_1 + (-t_2)$ will be written as $t_1 - t_2$, and (t+ . . . + t) (where t is repeated k times) will be written as kt. k is called a coefficient of t. The symbols ≤, ≥, and > will also be used. These are all definable in terms of < and + alone. The relationship | defined below will also be used. Note that | can be expressed using ∃ and =, as seen in Definition 1.

Definition 1: (divisibility relationship |)

$$c|t \triangleq \exists x[cx=t]$$

where t is a term, c is a positive constant, and x is a variable that does not appear in t. □

$c|t$ means that t can be divided by c.

2.2. Simplification of divisibility relationship

We present three theorems that are used to simplify the divisibility relationship.

Theorem 1: Let c and d be positive integers, and t be a term; then,

$dc|dt \Leftrightarrow c|t$. □

Theorem 2: Let c be a positive integer, d be an integer, t be a term, and y be a variable or an integer; then,

$c|cdy + t \Leftrightarrow c|t$. □

Theorem 3: Let t be a term; then,

$1|t \Leftrightarrow TRUE$. □

Theorem 1 signifies simplification by reduction. Theorem 2 shows simplification by elimination of a term that is a multiple of the divisor. Theorem 3 represents simplification by replacing a relationship divisible by 1 with *TRUE*.

2.3. Definition of EPP-sentence

An EPP-sentence is a P-sentence in prenex-normal form bounded only by existential quantifiers. The satisfiability of formula A on PA, whose variables are all free, can be determined by determining the truth of EPP-sentence B, which is made by binding all variables in A by existential quantifiers [14]. Here, A is unsatisfiable if and only if B is false.

2.4. Determination of truth of EPP-sentence

In this section, we describe a procedure for determining the truth of an EPP-sentence by quantifier elimination according to Ref. 11.

2.4.1. Quantifier elimination

The procedure of quantifier elimination is one in which, given a formula L^+ of the form $\exists xF(x)$ where $F(x)$ is quantifier free, returns a formula F' of L^+ which is equivalent to $\exists\, xF(x)$, but which contains neither quantifiers nor x. F' is expressed by a disjunction of several formulas F_0' of the form $\bigvee_{x_t=1}^{\delta}(F''(x_t) \wedge d|A(x_t))$. $F''(x)$ is obtained from $F(x)$ by performing translations so as to make the coefficients of x have the same value. The number of atoms in $F''(x)$ is equal to that of $F(x)$. Furthermore, $A(x_t)$ is a term and d is an integer. Also, δ is obtained from the least common multiple (LCM) of x's coefficients in $F(x)$, and so on. The number of the F_0' in F' is at most the same as the number of atoms in $F(x)$.

In F_0', x_t is a variable whose scope of value is restricted. In the following, we call this variable a scope-restricted variable. Refer to Refs. 11 and 14 for details of quantifier elimination.

2.4.2. Elimination of Quantifiers

In order to determine the truth of the EPP-sentence A, we first apply quantifier elimination to the innermost formula, of form $\exists\, xF(x)$, among the partial formulas of A, and transform it to an equivalent formula that does not contain quantifiers. Then, we apply quantifier elimination to a formula the next level. We repeat this procedure until no quantifier remains. Here, let A be $\exists x1$ . . . $\exists xnF$ $x_1, \ldots, x_n$); then, by eliminating all the quantities within A, A is transformed to a disjunction of several formulas as follows:

$$\bigvee_{t_1}^{\delta_1} \cdots \bigvee_{t_n}^{\delta_n} \left[F'(t_1, \cdots, t_n) \wedge \bigwedge_{j=1}^{n} \lambda_j \left| \sum_{k=1}^{n} \mu_{jk} t_k + \nu_j \right. \right] \quad (1)$$

The truth of this formula can be determined by replacing scope-restricted variables $t_1, t_2, \ldots, t_n$ in

$$F'(t_1,\cdots,t_n) \ \wedge \ \bigwedge_{j=1}^{n} \lambda_j \Big| \sum_{k=1}^{n} \mu_{jk} t_k + \nu_j \qquad (2)$$

with their values and interpreting it. EPP-sentence *A* is true if and only if there exists at least one transformed formula that is true.

2.4.3. Truth determination by assigning values

In determining the truth of formula (1) of Section 2.4.2, the amount of computation needed for interpretation of each formula can be reduced by the method shown below. First, we simplify formula (1) using Theorems 1, 2, and 3 if they can be applied. Next, we obtain a combination of values for scope-restricted variables that makes the necessary condition for formula (1)

$$\bigvee_{t_1}^{\delta_1} \cdots \bigvee_{t_n}^{\delta_n} \left[\bigwedge_{j=1}^{n} \lambda_j \Big| \sum_{k=1}^{n} \mu_{jk} t_k + \nu_j \right] \qquad (3)$$

true. That is, we find a combination of values which makes the conjunction

$$\bigvee_{t_1}^{\delta_1} \cdots \bigvee_{t_n}^{\delta_n} \qquad (4)$$

of all the atoms that express the divisibility relationship true within the scope

$$\bigwedge_{j=1}^{n} \lambda_j \Big| \sum_{k=1}^{n} \mu_{jk} t_k + \nu_j \qquad (5)$$

of variables. Then we assign those values to $F'(t_1, \ldots, t_n)$

$$F'(t_1,\cdots,t_n) \qquad (6)$$

and determine the truth of formula (6). Formula (1) is true if there is at least one combination that makes formula (6) true.

3. Detection of IFP

In order to decide whether a path is an IFP or not, we first obtain the path conditions by symbolic execution. A path condition is a condition that must be satisfied for the input variables to execute a program along a path. Next, we determine the satisfiability of the path condition with a theorem prover. If the path condition is unsatisfiable, the path is decided to be an IFP [1].

3.1. Calculation of path conditions

We previously proposed a method to compute path conditions [3] by interpreting a directed graph of a program (called a path dependence flow graph [3, 6]) based on a demand-driven execution model. In this method, the path conditions can be constructed efficiently because the results of partial computations are shared and only indispensable computations are performed.

There are cases where computation of path conditions does not terminate because of recursive calls when a program contains procedure calls. Therefore, we do not perform interprocedural analysis. Rather, we perform only analyses closed within procedures. As the result, function calls remain in the path conditions when the computation of path conditions is finished. We will describe a procedure for formulas with function calls in Section 3.4. If a program contains loops, computation of path conditions may not terminate as in the case of procedure calls. However, we propose a method to transform a loop into a procedure call. Computation of path conditions always terminates when procedure calls are included. Therefore, it is guaranteed that the computation terminates even if a program contains loops [3].

3.2. Satisfiability determination of path conditions

Since a path condition is a formula of the first-order predicate calculus, the problem of satisfiability of path conditions results in a validity problem of the first-order predicate calculus. However, such a validity problem is unsolvable in general. Therefore, it is not always possible to judge the satisfiability of path conditions [14]. However, if certain paths are known to be IFPs when performing analysis along a path, the paths can be excluded from the target of analysis and the overall precision of analysis improves. For instance, if a def-use relationship [17] of a program holds only along a certain path and the path is an IFP, then the referencing variable is not influenced by the defining variable for any input of execution. For that reason, program slices can be constructed more precisely if some paths are determined to be IFPs [8]. From this fact, semantic errors that may occur when two variants of a program are merged can be detected accurately [7]. Furthermore, the precision of analysis is improved by finding as many IFPs as possible. Thus, it is sufficient that as many path conditions as possible be determined to be unsatisfiable.

Therefore, we first choose a class whose satisfiability can be determined as widely as possible, and exactly determine the satisfiability of path conditions of that class. For path conditions outside the class, we determine their satisfiability using a conservative approximation [3]. In the following, we call a path detected to be an IFP as a detected IFP (DIFP), and we call those not so detected feasible path candidates (FPC).

3.3. Satisfiability determination of path conditions using determination of the truth of EPP-sentences

We have proposed a method in which a class of formulas on a PA is used as a class whose satisfiability can be determined [3]. The satisfiability of such formulas can be determined by determining the truth of EPP-sentences as stated in Section 2.3.

3.4. Satisfiability determination by conservative approximation

A path condition generally contains function call terms, operations, and variables of various types not permitted in PA. If the path condition does not belong to a class of formulas on PA, we transform it into a formula on PA by replacing a term not permitted in PA with an integer-type variable, as described in the next section. If different terms can have different values, we replace them with different variables.

After this transformation, the satisfiability of path condition A can be judged by the conservative approximation. That is, it can be guaranteed that A is unsatisfiable if transformed formula B is unsatisfiable. The reason is that if B, which contains replaced variables, is unsatisfiable, then A is unsatisfiable whatever value is taken by the terms before replacement.

3.5. Transformation from path condition to formula on PA

We now present a procedure to transform a path condition that is not a formula on PA into a formula on PA using the conservative approximation.

(1) First, different results may be obtained owing to rounding-up error in the evaluation of a term that contains real variables or constants, if the operations are performed in different order [4]. Therefore, if there are real variables or constants, we transform each one into a new integer variable. For instance, the term $f + g$, which has real variable f and real constant g, is replaced by a new integer variable h.

(2) Next, we transform variables and constants not of integer type into integer variables as shown in Ref. 4. Concretely, in the case of structured data such as arrays, each element is transformed into an integer variable. For instance, term $a[i] + b[i]$ is transformed into $p + q$ on PA by replacing $a[i]$ with integer variable p and $b[i]$ with integer variable q.

(3) Furthermore, we transform a term with operations not permitted by PA into an integer variable as shown in Ref. 3. For instance, term $f(a) + b$ and term $f(a) + c$ are transformed into $x + b$ and $x + c$, respectively, if the scope of variable a is the same. Here, $f(a)$ is a function application term.

4. High-Speed Determination of Truth of EPP-Sentences

According to experiments with a prototype system that employs a previous determination method, the computation time necessary for determination increases rapidly as the EPP-sentence becomes more complex [10]. In this section, we propose a high-speed method of determining the truth of EPP-sentences.

4.1. Amount of computation required for determining truth of EPP-sentences

The amount of computation required for determining the truth of EPP-sentence A increases with the number n of atoms in A, the number v of variables, and the absolute value u of the variables' coefficients [14]. The reason is that the number p of formulas (1), the number q of atoms in formula (1), and the maximum value δ_i of scope-restricted variables in formula (1) increase as n, v, and u increase.

Because the problem of determining the truth of EPP-sentences is NP-complete, the computation time cannot be reduced when v is not fixed and large. On the other hand, if v is set to a value not less than 2, there is a possibility that the truth of an EPP-sentence can be evaluated within a deterministic polynomial time for n and u, because it is undetermined whether the truth of the EPP-sentence can be determined within a deterministic polynomial time. Therefore, we consider the case where u is fixed in the following.

If a quantifier is removed from EPP-sentence A, the number of F_0' in Section 2.4.1 becomes, at most, n, and the number of atoms in F_0' becomes at most $n + 1$. Therefore, even if all the quantifiers are removed from A by v applications of quantifier elimination, p and q are expressed in a polynomial order of n.

However, the upper bound of δ, which is generated by eliminating one quantifier in A, increases as u^n even in simple cases where A has no divisible atom. The reasons are that δ is the LCM of the coefficients of the variables in question, δ becomes large when the coefficients are equal to or close to u and are mutually indivisible, and the maximum number of coefficients is the number u of atoms. This order signifies that the number of combinations of values of scope-restricted variables increases explosively as n increases.

In this section, we aim to reduce computation time when the absolute values of the coefficients and the number of atoms increase, and we propose an efficient determina-

tion method for the truth of EPP-sentences, which reduces the number of combinations of values to be examined when the number of variables is fixed.

4.2. Cooper's suggestion

Cooper presents in Ref. 11 a guideline to find a combination of values that makes formula (3) true. First, in formula (3), for the conjunction of atoms that express the divisibility relationship, we triangulate matrix (μ, jk) (called the coefficient matrix), which expresses the coefficients of variables that appear in each atom, by using a method similar to Gaussian elimination. Next, we narrow down the combinations of values of scope-restricted variables in the same manner as back-substitution. Here we use the following theorem in number theory for the triangulation and narrowing down by back-substitution.

Theorem 4: Let t be a variable on a set of integers Z, and let A be a set of terms that does not include t. Also, let $GCD(k, l)$ be the greatest common divisor of integer k and l. Then, if $m_j \in Z$, $a_j \in Z$, and $b_j \in A$ for $j = 1, 2$,

$$m_1 | a_1 t + b_1 \wedge m_2 | a_2 t + b_2$$
$$\Leftrightarrow m_1 m_2 | dt + b_1 p_1 m_2 + b_2 p_2 m_1 \wedge d | a_2 b_1 - a_1 b_2$$

where $d = GCD(a_1 m 2, a_2 m 1)$ and $p_1 a_1 m_2 + p_2 a_2 m_1 = d$. □

p_1 and p_2 are obtained as by-products when Euclid's algorithm is used to obtain the greatest common divisor.

Theorem 4 shows that there is a transformation such that variable t included in two atoms is included in only one atom.

Theorem 5: Let t be a variable on Z. If $m \in Z$, $a \in Z$, and $b \in Z$, then $m | at + b$ has a solution for $t \Leftrightarrow d | b$.

Let i be an arbitrary integer, then the solution is $t = -p(b/d) + i(m/d)$.

Here, $d = GCD(A, m)$, and $pa + qm = d$. □

Theorem 5 shows the condition for a variable to have a solution when the variable appears in an atom that expresses the divisibility relationship, and the solution when the condition is satisfied.

The concrete procedure of finding the combination of values is as follows. First, triangulate the coefficient matrix using Theorem 4. Then, obtain solutions for the variables in each atom from Theorem 5 and the scope of the variables. Finally, obtain solutions for the other variables successively by back-substitution.

4.3. Determination by triangulation and back-substitution

When we solve a multivariable linear equation by Gaussian elimination, the solution is indefinite or impossible if the rank of the coefficient matrix of the equation is less than the degree of the matrix. On the other hand, when we search for combinations of values of variables that make formula (3) true, there are cases where back-substitution cannot be applied if the rank of the coefficient matrix is less than the degree according to Cooper's indication. We call such matrices lower rank matrices in the following. In this section, we propose a method that allows triangulation and back-substitution to be applied even to lower rank matrices Furthermore, we describe a concrete procedure for finding combinations of values of variables that make formula (3) true. In this procedure, if the triangulated coefficient matrix produced by Cooper's method is a lower rank matrix, it is transformed into a matrix to which back-substitution is applicable by solution of a multivariable linear indefinite equation [16].

4.3.1. Triangulation using a theorem in number theory

We now describe the procedure for triangulating the coefficient matrix. This procedure transforms the lower triangular elements of the coefficient matrix to 0 in the same way as Gaussian elimination. Theorem 4 is used to transform each element to 0. Furthermore, we simplify the formula (if it can be simplified) in order to make the computation efficient, as in Section 2.4.3. The concrete transformation method is described in Ref. 10.

Formula (5) is transformed into an equivalent formula by application of triangulation

$$\bigwedge_{j=1}^{r} \lambda'_j \left| \sum_{k=j}^{n} \mu'_{jk} t_k + \nu'_j \right. \tag{7}$$

which has a triangulated coefficient matrix. Here, $\lambda'_j \in Z$, $\mu'_{jk} \in Z$, and $\nu'_j \in A$ with $1 \leqq j \leqq n$ and $1 \leqq k \leqq n$. Notice that atoms beginning with λ'_j contain $t_j, t_{j+1}, \ldots, t_n$. In this sense, the coefficient matrix is said to be triangulated. r is the rank of coefficient matrix (μ'_{jk}), $1 \leqq r \leqq n$. Also notice that for $l (r < l \leqq n)$, all elements of the l-th row are 0.

4.3.2. Triangulation of lower rank matrix

When we obtain the value of variable $t_k (1 \leqq k \leqq n)$ by back-substitution, the value of variable t_l must have been obtained if the k-th row and l-th column element μ'_{kl} is not 0 for any $l (k < l \leqq n)$. In order for the value of variable t_l to be obtained, the l-th row, l-th column element μ'_{ll} must be nonzero. In other words, μ'_{ll} must not be 0 if μ'_{kl} is nonzero for any $l (k < l \leqq n)$ in order to obtain the value of t_k by back-substitution. However, there are cases in which this condition is not satisfied in lower rank matrices. Thus, there

are cases where back-substitution cannot be applied in lower rank matrices.

Incidentally, coefficient matrix (μ'_{jk}) triangulated by the method in Section 4.3.1 may be a lower rank matrix. Thus, there are cases in which back-substitution cannot be applied to coefficient matrix (μ'_{jk}).

To solve the above problem, we describe a procedure for transforming a coefficient matrix to which back-substitution cannot be applied into one to which back-substitution can be applied. This procedure transforms the row elements in a coefficient matrix whose diagonal element is 0 and makes the diagonal element nonzero by using Theorem 6 described below.

Theorem 6: Let t_r be a variable on a set of integers Z; then,

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

where $m \in Z$, $a_r \in Z$, $b \in Z$, $n \geqq 2$, $1 \leqq r \leqq n$, $1 < l \leqq n$.
Proof: Given in the appendix. □

An atom that contains variables $t_1, t_2, \ldots, t_n$ can be transformed into a conjunction of the atom itself and an atom that contains $t_l, t_{l+1}, \ldots, t_n$ by using Theorem 6. Here, l is an arbitrary value and $1 < l \leqq n$.

In this procedure, if back-substitution cannot be applied to a coefficient matrix because diagonal element μ'_{kk} is 0, the value of μ'_{kk} can be transformed to a nonzero value by using Theorem 6. Furthermore, the elements of rows other than the k-th row and the lower triangular elements of the k-th row are invariant. Thus, by using this procedure, it is always possible to transform matrix (μ'_{kl}), to which back-substitution cannot be applied, into one to which it can be applied.

4.3.3. Back-substitution

By applying the procedures shown in Sections 4.3.1 and 4.3.2, formula (5) is transformed to

$$\bigwedge_{j=1}^{n} \lambda''_j \Bigg| \sum_{k=j}^{n} \mu''_{jk} t_k + \nu''_j \tag{8}$$

Here, notice that the coefficient matrix (μ'_{jk}) is triangulated and back-substitution can be applied. Therefore, we obtain a combination of values of scope-restricted variables that makes formula (8) true by applying back-substitution in accordance with Theorem 5. The concrete procedure is described in Ref. 10.

5. Example of IFP Detection

We present two examples of IFP detection from a program.

5.1. Example without lower rank matrix

We present an example that detects IFPs from the program shown in Fig. 1. We assume that the values of u, v, and w are not changed in the parts shown by . . . . In this example, the coefficient matrix used in determination of the truth of EPP-sentences is not a lower rank matrix.

This program contains three "if" statements and 8 ($= 2^3$) paths. In this section, we turn our attention to path P_{ttf}, where the conditions of the first and the second "if" statements become true, and the condition of the third "if" statement becomes false for simplicity of discussion. In order to determine whether P_{ttf} is an IFP, we first determine the satisfiability of the path condition

$$2x+3y<300 \wedge 3x+y>300 \wedge 7y-x>201 \tag{9}$$

of P_{ttf}. That is, we determine the truth of EPP-sentence

$$\exists x \exists y [2x+3y<300 \wedge 3x+y>300 \wedge 7y-x>201] \tag{10}$$

This EPP-sentence is concluded to be false as shown in Ref. 10, and P_{ttf} is decided to be a DIFP.

In this example, back-substitution can be applied to the triangulated coefficient matrix. Thus, the procedure described in Section 4.3.2 is not needed. Furthermore, as shown in Ref. 10, 21 × 4998 combinations of values are needed for this example in the previous method for deter-

```
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 {...}
    ...
}
```

Fig. 1. Sample program 1.

mining the truth of P-sentences. However, only three combinations of values are needed in a determination that uses triangulation.

5.2. Example with lower rank matrix

We present an example of the detection of IFPs from the program shown in Fig. 2. We assume that the values of s, t, u, and v are not changed in the part of program shown by . . . , as was the case in Fig. 1. In this example, the coefficient matrix for determining the truth of EPP-sentence becomes a lower rank matrix.

This program includes four "if" statements. In this section, for simplicity, we turn our attention to path P_T, where the condition of each "if" statement becomes true. Whether P_T is an IFP or not can be determined as follows. First, we determine the truth of the EPP-sentence

$$\exists x \exists y \exists z [-12x+y+z>-1 \ \wedge \ y+z>-2 \ \wedge \ x-y+z>-5 \ \wedge \ x+4y-5z>0] \quad (11)$$

in order to determine the satisfiability of the path condition of P_T. This EPP-sentence is determined to be true as shown below. Therefore, P_T is determined as FPC.

Formula (11) is transformed to

$$\bigvee_{t_Z=1}^{5} \bigvee_{t_Y=1}^{180} \bigvee_{t_X=1}^{59400}$$

```
main(x,y,z)
int x,y,z;
{
    int s,t,u,v;
    t = y + z + 2;
    s = t - 12 * x - 1;
    u = x - y + z + 5;
    v = x + 4 * y - 5 * z;
    ...
    if (s > 0) then {...} else {...}
    if (t > 0) then {...} else {...}
    if (u > 0) then {...} else {...}
    if (v > 0) then {...} else {...}
    ...
}
```

Fig. 2. Sample program 2.

$$[129600t_X+3240t_Y+t_Z < 2986200 \ \wedge \ 11880 \mid t_X+6t_Y-1080t_Z-16200 \ \wedge \ 9900 \mid t_X-5t_Y+900t_Z-6300 \ \wedge \ 59400 \mid t_X+270t_Y+10800t_Z-253800] \quad (12)$$

by eliminating quantifiers using the procedure shown in Section 2.4.2. As described in Section 4.3.1, if we eliminate t_x from two atoms in divisibility relationships of formula (12) using Theorem 4, the conjunction of divisibility relationships can be expressed as

$$59400 \mid 10800t_Z-60t_Y+t_X+43200 \ \wedge \ 180 \mid 180t_Z-t_Y+9900 \ \wedge \ 180 \mid t_Y-900 \quad (13)$$

We simplify it using Theorem 2 as

$$59400 \mid 10800t_Z-60t_Y+t_X+43200 \ \wedge \ 180 \mid t_Y \quad (14)$$

Here, the coefficient matrix becomes

$$\begin{pmatrix} 10800 & -60 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

The third-row, third-column diagonal element of this matrix is 0. Therefore, back-substitution cannot be applied.

However, as shown in Section 4.3.2, from Theorem 6,

$$59400 \mid 10800t_Z-60t_Y+t_X+43200 \Leftrightarrow 59400 \mid 10800t_Z-60t_Y+t_X+43200 \ \wedge \ 60 \mid t_X+43200 \quad (15)$$

Therefore, the coefficient matrix can be transformed to

$$\begin{pmatrix} 10800 & -60 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

and back-substitution is applicable to this matrix. We determine the truth of formula (11) using back-substitution as described in Section 4.3.3.

6. Discussion

We will discuss the computation time required by the high-speed method of determining the truth of EPP-sentences. Additionally, we will discuss conditions for the high-speed method to be effective. We will also discuss the class of EPP-sentences whose truth can be determined practically, and the effect of IFP detection, as well as

processing of lower rank matrices. Lastly, we will describe an evaluation by a prototype system.

6.1. Discussion of computation time

We consider the reduction of computation time by the high-speed method in this section. First, we consider the amount of computation required to find a combination of values that make formula (1) true, using a previous method that does not use triangulation of the coefficient matrix. $\prod_{j=1}^{n} \delta_j$ assignment processing steps are required in formula (1). On the other hand, in the high-speed method, the variable's occurrences are triangulated as in formula (8). Therefore, the candidate values of the scope-restricted variables can be narrowed down by Theorem 5. Stated simply, each value is in every (m/d) as stated in Theorem 5. Thus, where δ_i assignments for a certain variable t_i were required in the previous method, only δ_i/α_i assignments are required in the high-speed method, where α_i is some integer. That is, only $\prod_{i=1}^{n} \delta_i/\alpha_i$ assignments are needed.

Put concretely, 21×4988 assignments are needed in the previous method in the example of Section 5.1, but only three assignments are needed in the high-speed method.

6.2. Effect of high-speed method

The method of determination of the truth of EPP-sentences is widely utilized [12] to verify specifications in hardware design [18], communications protocol design [19], and general program design [20]. In many cases in these design environments, the absolute value of the coefficient of each variable is 1, and the values are not large in general. In that case, $\delta_j (1 \leqq j \leqq n)$ in formula (1) does not become so large and the effect of the high-speed method is not so significant. In contrast, the computation for triangulation will constitute overhead.

However, the target P-sentence is constructed from the path condition in IFP detection, and the coefficients of the variables can take arbitrary values in the path conditions. Therefore, the values of the coefficients may become large, and the coefficients of the variables in the EPP-sentences may become large. Thus, the value of $\delta_j (1 \leqq j \leqq n)$ becomes large in general, and the reduction in computation achieved by triangulation becomes larger than the overhead of triangulation. Therefore, the speedup method is effective when the coefficients of variables are large.

6.3. A class of EPP-sentences whose truth is practically determinable

We consider a class of EPP-sentences whose truth is practically determinable by the high-speed method of determination of the truth of EPP-sentences.

First, we analyze the method of high-speed determination of the truth of EPP-sentences using a uniform cost criterion in the random access machine (RAM) for the machine model, and we show that the truth of two-variable EPP-sentences can be determined in a deterministic polynomial time [14]. Therefore, the truth of two-variable EPP-sentences can be determined within a practical time, even if the number of atoms and absolute values of coefficients are large.

Next, if the number of variables is greater than two but not very large, and the number of atoms is not very large, there is a possibility of determination within a practical time.

6.4. Effect in IFP detection

In this article, as stated in Section 4.1, we have aimed at accelerating the determination of the truth of EPP-sentences where the number of variables is fixed. In Section 4.3, we proposed a method for high-speed determination of the truth of EPP-sentences. In addition, using the proposed determination method, we described a class of EPP-sentences whose truth is practically determinable in Section 6.3.

In EPP-sentence A, whose truth is to be determined in IFP detection, the number n of atoms corresponds to the number of "if" statements along the path which is the target of IFP determination, and the number v of variables corresponds to the number of variables necessary for computation of the conditions in each "if" statement. Furthermore, the maximum absolute value of the coefficients in A corresponds to the maximum value of the coefficients of the variables needed for computation of the conditions. In real IFP detection, n, v, and u have the possibility of becoming large.

In previous methods of determination of the truth of P-sentences, the time required for determination increases rapidly if n, v, or u becomes large. For this reason, the class of A whose truth was determinable was small. However, in the high-speed method of determination of the truth of EPP-sentences, determination is possible within a practical time if v is small, even if n and u are large. The proposed method makes it practically possible to determine the truth of EPP-sentences in such a region.

There are cases where even the proposed high-speed determination method takes too much time, e.g., when v is large. A new high-speed technique applicable to as many instances as possible is necessary to make the truth of A practically determinable in those cases. Therefore, we proposed a technique [15] that partitions A into multiple EPP-sentences that have small numbers of variables and atoms, by applying techniques such as program slicing [6]. In this case, if the truth of each EPP-sentence generated by parti-

tioning is practically determinable, the truth of A is also determinable.

6.5. Processing of lower rank matrices

In the case of the lower rank matrices discussed in Section 4.3.2, back-substitution cannot be performed if the diagonal elements are 0. Therefore, we described a method of triangulating it using Theorem 6 in Section 4.3.2. Incidentally, there is a way to make the determination without triangulation, by assigning all possible values of the scope-restricted variables to formula (6). However, the range of scope-restricted variables is wide in general. If we perform the triangulation described in Section 4.3.2, back-substitution becomes applicable and, moreover, the values of the scope-restricted variables can be narrowed down for the sake of back-substitution. The overhead of triangulation is not very large. Therefore, it is more efficient to use the method described in Section 4.3.2 to reduce the computation time.

6.6. Evaluation by prototype system

In this section, we describe the results of experiments using prototype systems. We made a comparison using a prototype system that implemented the previous algorithm and a prototype system that implemented the high-speed algorithm. In this comparison, we use a Sun SPARC Station20 HS21 (125 MHz CPU clock, 96 MB memory) and measured the CPU time.

First, the truth of EPP-sentence (10) was determined to be false in 6 ms by the high-speed method, but in 47 s by the previous method. This is about a 7000-fold improvement in speed.

EPP-sentence

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

was determined to be false in 5 ms by the high-speed method, but 8 s were required by the previous method. This is about a 1000-fold improvement in speed.

Furthermore, in order to study the general effect of the high-speed method, in the EPP-sentence

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

we varied the values of a and b within the ranges $0 \leqq a \leqq 19$ and $1 \leqq b \leqq 9$ and compared the computation time of the previous method and the high-speed method. The determination was completed in 4–7 ms in the high-speed method. The computation time for the previous method is shown in Fig. 3. In the previous method, the computation time increased rapidly as the absolute values of the coefficients of the variables in the P-sentences increased, but in the high-speed method, it did not increase greatly. The effect of the high-speed method can be confirmed by calculating the ratio of the computation time of the high-speed method to that of the previous method. If either a and b is fixed, the ratio alternately increases and decreases on the graph as the variable's coefficient increases. However, if the ratio is shown in logarithmic coordinates, a curve that passes near the maximal points can be drawn in the graph. This curve is expressed by pu^q, where p and q are constants and u is the value of coefficient a or b. This shows that the maximal points of the ratio increase in the order of u^q. Figure 4 shows an example in which coefficient b's value is fixed to 6 and 9. The ratio has its maximum at $(a, b) = (19, 9)$, where its value is 3 million.

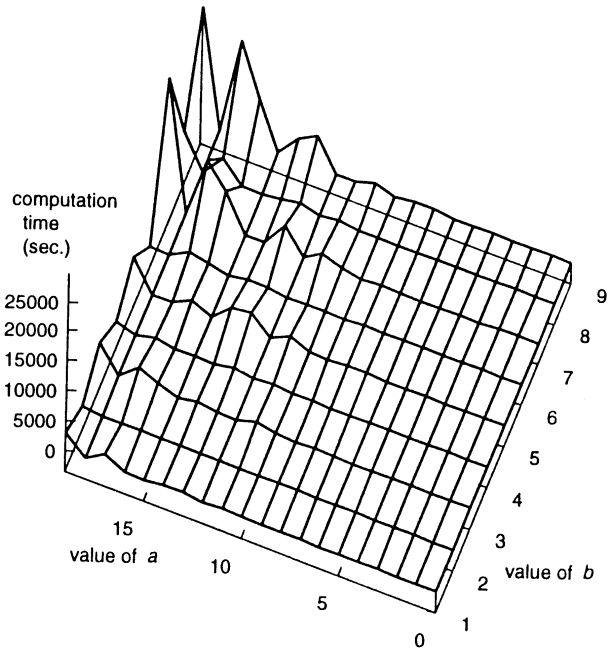

Fig. 3. Computation time using previous method.

7. Related Studies

We now discuss related studies of the determination of the satisfiability of path conditions in IFP detection, and of determination of the truth of EPP-sentences.

7.1. Satisfiability determination in IFP Detection

The conventional determinations of satisfiability of path conditions in IFP detection can be divided into two classes. One involves the application of the automatic theo-

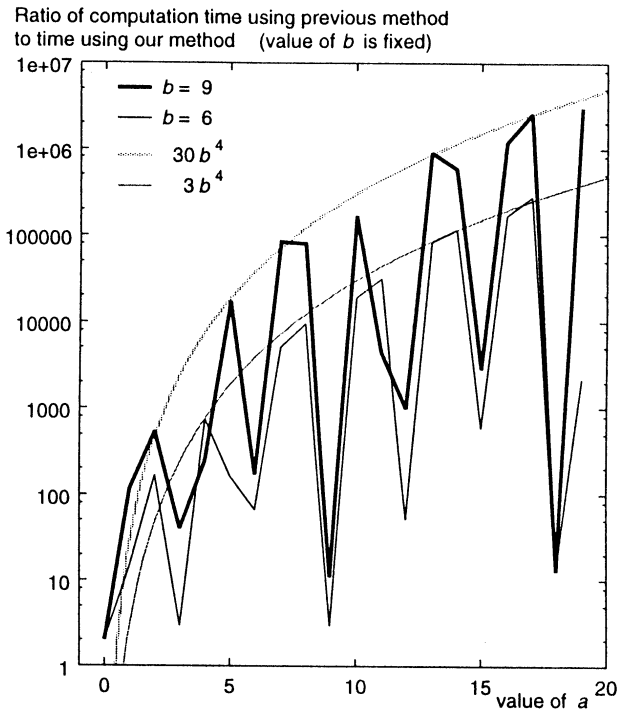

Fig. 4. Ratio of computation time using previous method to time using our method (value of b is fixed).

rem proving to the path conditions themselves [1, 4, 5], and the other restricts the class of target formulas and definitively determines the satisfiability of formulas in the class [2, 3].

The direct path condition determination method can determine satisfiability precisely for a wide variety of path conditions, but there are path conditions for which satisfiability cannot be determined. The reason is that the satisfiability problem of path conditions leads to a determination problem of first-order predicate calculus [10], but determination problems are unsolvable in general. Therefore, in this method, a path condition that can be determined to be unsatisfiable by the method for determination of the truth of EPP-sentences, cannot necessarily be determined unsatisfiable.

On the other hand, in the method that restricts the class of target formulas, the satisfiability of path conditions is determined by linear programming (LP), or by integer programming (IP) [2] if the path condition is constructed with only linear constraints. When LP is used, a path condition is determined to be unsatisfiable if there exists no integer solution that makes the path condition satisfiable. When IP is used, unsatisfiability is determined if there is no integer solution that makes the path condition satisfiable. In general, an integer solution does not necessarily exist even if there exists a real solution to linear constraints. Therefore, more path conditions can be determined to be unsatisfiable by the IP. Every formula in a class of formulas whose satisfiability can determined using the method of determination of the truth of EPP-sentences can also be determined by IP, as described in Ref. 14. In the next section, we will compare various methods that can determine the satisfiability of such formulas.

7.2. Method of determination of the truth of EPP-sentences

As shown in Section 7.1, formulas whose satisfiability can be determined by the method of determination of EPP-sentences can also be determined by IP. However, in the case of IP, the satisfiability problem for such formulas is NP-complete even if the number of variables is fixed [14]. On the other hand, if we use the method of determination of the truth of EPP-sentences, if the number of variables is fixed to a value greater than one, it is not known whether the truth of EPP-sentences can be determined within a deterministic polynomial time. Therefore, there is a possibility that the satisfiability of path conditions can be determined at high speed. Actually, we have shown that the satisfiability of two-variable EPP-sentences can be determined within a deterministic polynomial time in Section 6.3.

The Omega test is an algorithm for solving IP problems within a practical time, which is used in array data dependency tests, and so on [21]. Its prototype system is publicly available on the Internet by ftp. Furthermore, that system can determine the truth of EPP-sentences. However, Pugh et al. state [22] that there are cases where an extremely long time is required for determining the truth of EPP-sentences, and our experiment also confirms this fact. For example, determination of the truth of the EPP-sentence

$$\exists x \exists y \left[\bigwedge_{j=1}^{10} (\neg(x{=}2j)) \wedge \bigwedge_{j=1}^{10} (\neg(y{=}2j)) \right]$$

takes more than 10 min in a prototype system with the Omega test algorithm, but our prototype system makes the determination in less than 10 ms. Therefore, determination of the truth of EPP-sentences by the Omega test takes an enormous time in some cases, and thus it is hardly practical.

Cooper states in Ref. 11 that the number of combinations of values of scope-restricted variables can be reduced in determining the truth of EPP-sentences by triangulation of the coefficient matrix and back-substitution. However, he does not specify concrete procedures, such as a procedure when back-substitution cannot be applied.

Morioka et al. proposed a method of determining the truth of EPP-sentences rapidly by controlling the sequence

of variable elimination [23]. That method is especially effective when the absolute values of the coefficients of the EPP-sentences are small. On the other hand, in our high-speed method, the speedup becomes significant when the absolute values of coefficients are large.

8. Conclusions

In this article, we proposed an efficient method for determining the truth of Presburger sentences (EPP-sentences), bounded by only existential quantifiers, for use in IFP detection. In the proposed method, we first transform the EPP-sentence to a conjunction of divisibility relationships by quantifier elimination. Next, we triangulate the matrix (coefficient matrix) that expresses the coefficients of the variables that appear in the conjunction of divisibility relationships, using a method similar to Gaussian elimination. Furthermore, if back-substitution cannot be applied to the triangulated coefficient matrix, we transform it using a method for solving one linear equation with two or three unknowns so that back-substitution can be applied to the coefficient matrix. Then we determine the truth of the EPP-sentence by applying back-substitution.

Our method is based on the method suggested by Cooper. However, we made Cooper's suggestion into a concrete method, and we made it clear that there are cases where back-substitution cannot be applied if the rank of the triangulated coefficient matrix is less than the degree of the matrix. We have proposed a transformation method for such cases, so that back-substitution can be applied to the coefficient matrix using the solution for multivariable linear indefinite equations. In this way, the truth of EPP-sentences is determinable irrespective of the rank of matrix.

In this article we have also determined quantitatively the reduction of the amount of computation by the proposed method. We have shown that in the proposed method, the speedup is greater when absolute values are large, as in IFP detection, than when the absolute values of coefficients of the EPP-sentences are small, as in the verification of communications protocols. We have also discussed a class of formulas whose truth is determinable by the proposed method and its effect for IFP detection. We have presented examples of reduction in computation time using a prototype system. We have confirmed a maximum three-million-fold speedup compared with the conventional method.

We are now experimenting with more examples in a prototype system in order to verify the efficiency of the proposed method. In the future, we plan to evaluate the computation time experimentally and to analyze the amount of computation by algorithms.

Acknowledgments. The authors used the Presburger processing system developed at Taniguchi Laboratory, Department of Information and Computer Science, Faculty of Engineering Science, Osaka University, in constructing a prototype system for high-speed determination of the truth of EPP-sentences. The authors express their thanks to Professor Ken'ichi Taniguchi and Associate Professor Teruo Higashino for enabling us to use the system, and to Research Associates Kozo Okano and Sumio Morioka for their cooperation in the use of the system. Also, the authors would like to express their appreciation of discussions with Masaki Ito, leader of Advanced Software Engineering Research Group, and researchers of the Global Computing Laboratory.

REFERENCES

1. King JC. Symbolic execution and program testing. Commun ACM 1976;19:385–394.
2. Clarke LA, Richardson DJ. Applications of symbolic evaluation. J Systems and Software 1985;5:13–35.
3. Naoi K, Takahashi N. Detection of infeasible paths with a path dependence flow graph. Trans IEICE 1993;J76-D-I:429–439.
4. Goldberg A, Wang TC, Zimmerman D. Application of feasible path analysis to program testing. Proc 1994 Int Symp Software Testing and Analysis (ISSTA) 1994:80–94.
5. Jasper R, Brennan M, Williamson K, Currier B, Zimmerman D. Test data generation and feasible path analysis. Proc 1994 Int Symp Software Testing and Analysis (ISSTA) 1994:95–107.
6. Naoi K, Takahashi N. Program slicing using a path dependence flow graph. Trans IEICE 1995;J78-D-I:607–621.
7. Naoi K, Takahashi N. Detecting an interference in a merged program with a semantic configuration manager. Proc 45th Ann Conv Inform Process Soc Jpn 1992;6T-1.
8. Naoi K, Takahashi N. Program analysis system with a path dependence flow graph. NTT R&D 1993;42: 1007–1016.
9. Naoi K. Takahashi N. Program analysis system with a path dependence flow graph. Proc 47th Ann Conv Inform Process Soc Jpn 1993;5D-9.
10. Naoi K, Takahashi N. Technique for reducing computation for detecting infeasible paths using Presburger arithmetic. Tech Rep IEICE 1995;SS95-19.
11. Cooper DC. Theorem proving in arithmetic without multiplication. Machine Intelligence 1972:91–99.

12. Higashino T, Kitamichi J, Taniguchi K. Presburger arithmetic and its application to program developments. Computer Software 1992;9:31–39.
13. Scarpellini B. Complexity of subcases of Presburger arithmetic. Trans Am Math Soc 1984;284:203–281.
14. Naoi K, Takahashi N. Complexity analysis of an algorithm with a triangulation technique for deciding the truth of a Presburger sentence involving two variables. Tech Rep IEICE 1996;COMP95-108.
15. Naoi K, Takahashi N. Computation reduction in infeasible path detection using a path dependence flow graph. Proc 48th Ann Conv Inform Process Soc Jpn 1994;6G-2.
16. Dickson LE. Linear diophantine equations and congruences. In: History of the theory of numbers. Vol II. New York: Chelsea Publishing; 1952. p 41–99.
17. Aho AV, Sethi R, Ullman JD. Compilers: Principles, techniques, and tools. Addison-Wesley; 1986.
18. Kitamichi J, Morioka S, Higashino T, Taniguchi K. Automatic correctness proof of the implementation of synchronous sequential circuits suing an algebraic approach. Proc 2nd Int Conf Theorem Provers Circuit Design (TPCD'94). In: Kropf T, Kumar R, editors. Lecture notes in computer science. Vol. 901. Berlin: Springer-Verlag; 1995. p 165–184.
19. Lu DA, Higashino T, Taniguchi K. An automatic derivation of test cases for LOTOS expressions with data parameters. Trans IEICE 1992;J75-B-I:734–743.
20. Morioka S, Okano K, Higashino T, Taniguchi K. Hierarchical design of stock management program using relational algebra and its correctness proof. Trans Inform Process Soc Jpn 1995;36:1091–1103.
21. Pugh W. A practical algorithm for exact array dependence analysis. Commun ACM 1992;35:102–114.
22. Pugh W, Wonnacott D. Static analysis of upper and lower bounds on dependences and parallelism. ACM Trans Prog Lang and Syst 1994;16:1248–1278.
23. Morioka S, Higashino T, Taniguchi K. An implementation of decision procedure for prenex normal form Presburger sentence with bounded only by existential quantifiers. Tech Rep IEICE 1995;SS95-18.

Appendix

1. Proof of Theorem 6.

$$m \mid \sum_{k=1}^{n} a_k t_k + b$$
$$\Leftrightarrow -mh + \sum_{k=1}^{n} a_k t_k + b = 0$$

(where h is a variable on Z)

$$\Leftrightarrow -mh + \sum_{k=1}^{n} a_k t_k + b = 0$$
$$\wedge -mh + a_1 t_1 = d_1 j_1$$
$$\wedge \bigwedge_{k=2}^{n-1} d_{k-1} j_{k-1} + a_k t_k = d_k j_k$$
$$\wedge d_{n-1} j_{n-1} + a_n t_n = -b$$
$$\wedge d_1 = GCD(m, a_1) \wedge \bigwedge_{k=2}^{n-1} d_k = GCD(d_{k-1}, a_k)$$

(where j_k is a variable on Z)

(This partitioning is known as a method for solving one linear equation with three or more unknowns [16].)

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

(where, $1 < l \leqq n$)

(This is because it is deduced by elimination of j_k from $d_{k-1} j_{k-1} + a_k t_k = d_k j_k$ and $d_{n-1} j_{n-1} + a_n t_n = -b$, where $1 \leqq k < n$.)

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

□

AUTHORS (from left to right)

Kuniaki Naoi (member) received his B.S. and M.S. degrees in applied physics from Waseda University in 1988 and 1990, respectively. He has been affiliated with Nippon Telegraph and Telephone Corporation since 1990. Currently, he is a research engineer at Global Computing Laboratory, NTT Software Laboratories, engaged in research of software re-engineering. He is a member of the Information Processing Society of Japan; Japan Society for Software Science and Technology; ACM; and IEEE Computer Society.

Naohisa Takahashi (member) received his B.S. and M.S. degrees in electrical engineering from the University of Electro-Communications in 1974 and 1976, respectively. He received his Ph.D. degree in computer science in 1987 from Tokyo Institute of Technology. In 1976, he joined NTT Electrical Communications Laboratories, and has been engaged in research in the areas of functionally distributed parallel computers, data-flow computing systems, functional programming, parallel programming debugger, software re-engineering, global network computing, and so on. Currently, he is a leader of Ultra-Parallel Programming Research Group, NTT Software Research Laboratory. He is a member of Information Processing Society of Japan; Japan Society for Software Science and Technology; ACM; and IEEE Computer Society.