A NEW METHOD FOR PROVING CERTAIN PRESBURGER FORMULAS

W. W. Bledsoe
The University of Texas
Austin, Texas, USA

Abstract

This paper describes a new method for determining the validity of certain formulas from Presburger Arithmetic, namely those with only universally quantified variables. To do this the notion of a Presburger formula, is generalized slightly to that of a quasi-linear formula.

This so called "sup-Inf" method seems particularly suited for proving certain verification conditions that arise from program validation, especially those in which "proof by cases" is required. It also eliminates the need for proof by enumeration, inherent in some methods described earlier In the literature, which sometimes require a search through a large number of consecutive Integers.

These algorithms have been programmed and used extensively as a part of an automatic theorem proving system.

1. Introduction

Presburger Arithmetic.

An expression Is said to be a formula in Presburger arithmetic if It Is a (well) formed algebraic expression, allowing only variables, Integer constants, addition and subtraction, the arithmetic relations < and =, the propositlonal calculus logical connectives, and quantification (either universal or existential). Constant multiplication is also allowed. See [3], Davis [2], and Cooper (1].

The Presburger algorithm is a decision procedure for Presburger arithmetic: Given a formula in Presburger arithmetic decide whether it is true or false. Two main steps are utilized in this process: (1) Elimination of quantifiers (and replacement of variables by constants) (11) Evaluation of the resulting formula (which has no variables) to determine its validity.

Cooper's method.

Cooper [1] utilizes such a procedure. For example his method would convert the theorem $\forall x(x < 1 \rightarrow x < 2)$ successively to

$$\sim \exists x(x < 1 \wedge 1 < x), \sim \bigvee_{j=1}^{1} [1+j < 1 \wedge 1+j],$$

$\sim [2 < 1 \wedge 1 < 2]$, which is easily verified as true.

Apparently Cooper's method can result in a search through a list of consecutive integers when the coefficients of x are not unity. For example, the theorem $\forall x(5x < 11 \rightarrow 7x < 16)$ is converted to $\sim \exists x(5x < 11 \wedge 15 < 7x)$, and

$$\sim \bigvee_{j=1}^{35} (75 < 75 + j \wedge 75 + j < 77 \wedge 75 + j \equiv 0 (\mathrm{mod}\ 35)),$$

which requires testing the expression for each of the integers, j= 1,2, . . .,35. In fairness to Cooper It should be stated here that such adverse examples apparently do not arise often in the applications he considers in [1].

Our method, and In more detail in [8], which Is descriDea oeiow, nanaies only the case of Presburger formulas with universally quantified variables. But it avoids the need for long searches through consecutive integers, and facilitates certain types of "proof by cases." It is not clear whether our methods can be extended to handle all Presburger formulas, with both universal and existential quantification.

Presburger arithmetic has many applications In the field of proving assertions about computer programs* There a theorem about the program, is required to be proved. Such theorems are often not originally stated as formulas in Presburger Arithmetic but are reduced to such as the proof proceeds. For example for an array A, we might be given the theorem: $\forall j[j \leq 4 \wedge \forall k(k \leq 5 \rightarrow A[k] \leq A[k+1]) \rightarrow A[j] \leq A[j+1]]$*

Backchalnlng on the second hypothesis generates the subgoal $(j \leq 4 \rightarrow j \leq 5)$ which is a formula in Presburger Arithmetic. Notice that j is a universally quantified variable or free variable and hence must be treated as a skolem constant in the proof. Many applications result In Presburger formulas with only universally quantified variables. In this paper we describe a procedure, the sup-inf method, for deciding the validity of such formulas.

First let us note that the above formula $(j < 4 \rightarrow j \leq 5)$ Is easily verified by adding the negation of the conclusion to the hypothesis, getting $(j \leq 4 \wedge 6 \leq j)$ and then combining to get the contradiction $(6 \leq j \leq 4)$ Our procedure does essentially the same thing on this example. We now describe in Section 2 our procedure for determining the validity of universally quantified Presburger formulas. In Section 3 we define the pivotal algorithms SUP and INF and prove in Section 4 of [8] that they terminate with desirable outputs when applied to quasi-linear formulas. Several examples are liated in Section 3, and given In more detail in Section 5 of [8]. This method has been programmed and used extensively in a program verification system [6].

2. The Sup-Inf Procedure

An Example using the procedure.

We begin with an example and then outline the general procedure.

Let F be the theorem

$$(3)\quad (2x_2 + 3 \leq 5x_3 \wedge x_3 \leq x_1 - x_2 \wedge 3x_1 \leq 5 \rightarrow 2x_2 \leq 3)\ .$$

Notice that F has 3 (universally quantified)

The skolemization process will not be discussed here. These universally quantified variables (or skolem constants) will be referred to as "variables" in this paper.

variables x_1, x_2, x_3. We will negate F and convert it to the expression (8') below, which gives a range for each of these three x 's. (Expressions (5'), (7'), and (8') correspond to expressions (5), (7), and (8) given later in our general procedure).

We first obtain

$$(5') \quad (2x_2 + 3 \le 5x_3 \wedge x_3 \le x_1 - x_2 \wedge 3x_1 \le 5 \wedge 3 \le 2x_2 - 1)^2$$

as the negation of F, and then convert it to

$$(7') \quad (x_3 + x_2 \le x_1 < \infty) \wedge (0 \le x_1 \le \tfrac{5}{3}) \wedge (0 \le x_2 \le \tfrac{5}{2} x_3 - \tfrac{3}{2}) \wedge (0 \le x_2 \le x_1 - x_3) \wedge (2 \le x_2 < \infty) \wedge (\tfrac{2}{5} x_2 + \tfrac{3}{5} \le x_3 < \infty) \wedge (0 \le x_3 \le x_1 - x_2),$$

and finally to

$$(8') \quad (x_3 + x_2 \le x_1 \le \tfrac{5}{3}) \wedge (2 \le x_2 \le \min(\tfrac{5}{2} x_3 - \tfrac{3}{2}, x_1 - x_3)) \wedge (\tfrac{2}{5} x_2 + \tfrac{3}{5} \le x_3 \le x_1 - x_2) .$$

To show the invalidity of (8') we calculate a lower bound $\underline{x}_1$ for x_1 and an upper bound $\bar{x}_1$, and check that the interval $[\underline{x}_1, \bar{x}_1]$ contains no integer. For this example, $\underline{x}_1 = \frac{17}{5}$ and $\bar{x}_1 = \frac{5}{3}$, and since $\frac{5}{3} < \frac{17}{5}$ we are finished. $\underline{x}_1$ and $\bar{x}_1$ are computed by the algorithms INF and SUP given in Section 3.

Quasi-Linear Formulae.

The reader will observe that expression (8') is not a Presburger formula because it contains non-integer constants ($\frac{5}{3}, \frac{5}{2}, \frac{3}{2}$, etc.), and the symbol "min". We now relax the condition on the integer constants to allow any rational number as well as ∞ and $-\infty$. However, the variables (such as x_1 , x_2, x_3 in the above example) will represent only non-negative integers. We also allow the symbols "max" and "min". Such an extended Presburger formula will be called quasi-linear. It will be called "quasi-linear in L" if each of Its variables is a member of the set L.

We now describe our general procedure for determining the validity of such quasi-linear formulas.

Let F be a quasi-linear formula in the variables $x_1, x_2, \ldots, x_n$. We first want to convert $\sim F$ to the disjunctive form $(F_1 \vee F_2 \vee \ldots \vee F_p)$[3] where each of the F_i is a conjunction of the form

$$(a_1 \le x_1 \le b_1) \wedge \ldots \wedge (a_n \le x_n \le b_n) ,$$

and each a_j, b_j are quasi-linear expressions in the other x_k (but not in x_j). This is done as follows:

The Sup-Inf Procedure.

First, place $\sim F$ in disjunctive normal form $(G_1 \vee G_2 \vee \ldots \vee G_p)$ where each G_i is a disjunct of the form

$$\bigwedge_{i=1}^{m} (A_i \le B_i \wedge C_i = D_i)$$

and the A_i, B_i, C_i, D_i are quasi-linear expressions in $x_1, x_2, \ldots, x_n$. We eliminate the equalities in (4) by converting each $(C_i = D_i)$ into $(C_i \le D_i \wedge D_i \le C_i)$[4] so that each G_i has the form

$$(5) \quad \bigwedge_{i=1}^{m} (A_i \le B_i) .$$

Now each $(A_i \le B_i)$ is converted into a set of exactly n inequalities

$$(6) \quad (a_{i1} \le x_1 \le b_{i1}) \wedge \ldots \wedge (a_{in} \le x_n \le b_{in})$$

by "solving for"[5] each of the x_j occuring in A_i and B_i in terms of the other x_L. Thus the a_{ij} and b_{ij} appearing in (6) are expressions in the other x_L (but not x_j). If x_j does not occur in A_i or B_i then we put $a_{ij} = 0$, $b_{ij} = \cdot\cdot$.

So (5) is converted to

$$(7) \quad \bigwedge_{i=1}^{m} (a_{i1} \le x_1 \le b_{i1}) \wedge \ldots \wedge \bigwedge_{i=1}^{m} (a_{in} \le x_n \le b_{in})$$

and finally (7) is converted to

$$(8) \quad (a_1 \le x_1 \le b_1) \wedge \ldots \wedge (a_n \le x_n \le b_n) ,$$

where, for $k = 1, n$, $a_k = (\max a_{1k}\ a_{2k} \cdots a_{mk})$[6] , $b_k = (\min b_{1k}\ b_{2k} \cdots b_{mk})$. Thus, by this whole process we convert $\sim F$ to a disjunct

$$F_1 \vee F_2 \vee \ldots \vee F_L \vee \ldots F_p$$

2 How $3 < 2X_2\text{-}1$ is derived f r $2x_2 \not\le 3$ s explained in Section 3 of [8].

In our example above we had only one F_i This has been the case in most examples we nave tried so far.

4 In practice we do not always convert the equalities to inequalities, but rather use a "substitution of equals*' technique to gain efficiency. See [9].

In solving for x_1 in an expression like $\sup(x_1 + 2, x_1) \le \hat{x}_3$, we obtain two answers: x. $\le x_3 - 2$ and $x_1 \le x_3$ instead of one, as indicated in formula (6). However, this presents no difficulty in proceeding to (7) and (8).

The function MAX (See Sect. 3) is applied to the a.. . If the maximum is not immediately attainable then the symbol "max" is employed. Similarly for MIN. See for example, (7') and (8') above.

where each F_L has the form

(9) $(a_{L1} \le x_1 \le b_{L1}) \wedge \ldots \wedge (a_{Ln} \le x_n \le b_{Ln})$,

and the a_{Lk}, b_{Lk} are quasi-linear expressions in the x_1.

Now we determine that F is valid by showing that each F_L is false.

Since the a_{LK} and b_{LK} are usually expressions in the other x_i (as was the case in our example) it is not immediately obvious how one can test for the invalidity of (9). Our method (the "sup-inf" method) for doing this is simply to test whether the interval

$$[\inf_S x_k, \sup_S x_k] ,$$

contains no integer, for some k, k=l,2, ...,n, where sup. x_k and $\inf_S x_k$ are defined as follows:

Definitions.

If S is a set of inequalities of the form (9) and x_1', x_2',.....,x' are real numbers, then $(x_1', xb', \ldots, x')$ is said to satisfy S if each inequality in S becomes true when each symbol x_K is replaced by the number x_k'.

If A is a quasi-linear expression in $x_1, x_2, \ldots, x_n$. and x_1' x'_2 ,...,x' are real numbers then

$$A(x_1'/x_1, \ldots, x_n'/x_n)$$

denotes the number gotten from A by replacing each symbol x_k by the number x_k'

If A is a quasi-linear expression in $x_1, x_2, \ldots, x$, then $\sup_S$ A is defined to be the least upper bound of all numbers

$$A(x_1'/x_1, \ldots, x_n'/x_n) ,$$

where $(x'_1, x_2', \ldots, x_n')$ is a sequence of non-negative integers satisfying S. Similarly, $\inf_S$ A is the greatest lower bound of such numbers. When no confusion will arise we omit the subscript S and write sup A and inf A.

Thus the validity of F haa been reduced to determining sup. x. and inf. x_k , where S is the set of conjuncts of (9). We compute $\sup_S x_k$ and $\inf_s x_k$ by the algorithms SUP and INF given in Section 3.

In Section 4 of [8] we prove that if S is any set of inequalities of the form (9) where we assume only that the a_{Li}, b_{Lf} are quasi-linear in $x_1, \ldots, x$, then the outputs SUP (x_k, NIL) and INF (x_k, NIL) are numbers **(or possible $\pm \infty$)** with the property

$$INF(x_k, NIL) \le \inf_S x_k ,$$

$$\sup x_k \le SUP(x_k, NIL) .$$

Furthermore, we conjecture there that if the set S is in "natural form", in that it has been derived from a theorem F by the procedure described above, then equality hold in the above formula, i.e.,

$$INF(x_k, NIL) = \inf_S x_k ,$$

$$\sup_S x_k = SUP(x_k, NIL) .$$

Thus to show the validity of F we need only show that the interval

$$[INF(x_k, NIL), SUP(x_k, NIL)]$$

contains no integer for some k, k = 1,2,...,n.

Some Discussion of the Procedure.

This procedure for deciding the validity of a quasi-linear formula (and hence for a universally quantified Presburger formula) is called the sup-inf method. Of course it serves much the same purpose as the methods of Cooper in [1] and others. However, we feel that the sup-inf method has some advantages, especially for proving theorems arising from program validation.

One such advantage is that a hypothesis, such as

$$(2x_2 + 3 \le 5x_3 \wedge x_3 \le x_1 - x_2 \wedge 3x_1 \le 5)$$

from our earlier example (3), can be stored in the concise form

$$(10) \quad (x_3 + x_2 \le x_1 \le \tfrac{5}{3}) \wedge (0 \le x_2 \le \min(\tfrac{5}{2} x_3 - \tfrac{3}{2}, x_1 - x_3)) \wedge (\tfrac{2}{5} x_2 + \tfrac{3}{5} \le x_3 \le x_1 - x_2) ,$$

to be used to establish various conclusions as required. Thus if we desire to establish $(2x_2 < 3)$ we need only update (10) with its negation $(3 < 2x_2\text{-}1)$ to get (8') and then show that (8') is invalid. Also, using this same hypothesis (10) we might (later) be required to prove another conclusion, which itself has a hypothesis, such as

$$(11) \quad (x_3 \le 5x_2 \rightarrow x_3 \le 8) .$$

In this case $(x_3 \le 5x_2)$ is used to update (8') getting

$$(x_3 + x_2 \le x_1 \le \tfrac{5}{3}) \wedge (\max(2, \tfrac{x_3}{5}) \le x_2 \le \min(\tfrac{5}{2} x_3 - \tfrac{3}{2}, x_1 - x_3)) \wedge (\tfrac{2}{5} x_2 + \tfrac{3}{5}) \le x_3 \le \min(x_1 - x_2), 5x_2)) ,$$

which is used to prove $x_3 < 8$.

Also as mentioned earlier it avoids proof by searches through long lists of integers.

While these arguments have merit, they are not our main reason for prefering the sup-inf

1 This conjecture has recently been proved correct by Shotak [10].

method, which is our desire to efficiently handle certain "proof by cases." This is explained in [8, pp. 11-15), and in [7).

3. Algorithms

Here we describe the pivotol algorithms SUP and INF.

If A and B are quasi-linear expressions in L (see definition in Section 1), then so also are (A+B), max(A,B), min(A,B), and r * A, where r is a rational number. We can also divide a quasi-linear expression A by a non-zero rational number r by multiplying by its inverse, i.e., **1/r * A.**

Let S be a set of inequalities of the form $a \leq x_j \leq b$ where a and b are quasi-linear in $x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_n$. Recall the definitions of $\sup_S x_j$ and $\inf_S x_j$ given in Section 2. We now give algorithms for computing $\sup_S x_i$ and $\inf_S x_i$ for a given S. S will be assumed to be fixed throughout the remainder of this section.

Definition. **MAX(x,y) is a function which returns y if $x \leq y$, x if $y < x$, and ("max" x y) otherwise. Similarly for MIN(x,y). If J is a variable, then S contains one (and only one) inequality of the form $a \leq J \leq b$, which represents knowledge about J. If a = 0 and $b = +\infty$ then nothing is stated about J except that it represents a non-negative integer.**

The notation LOWER(S,J) and UPPER(S,J) is used to denote these lower and upper bounds on J. For example, if $(2 \leq j \leq 3\text{-}K)$ is in S, then LOWER(S,J) is 2, and UPPER(S,J) is 3-K.

SIMP is an algorithm that puts expressions in canonical form. See (8,7]. All outputs from the algorithms SUP, SUPP, INF, INFF, are automatically simplified by applying the algorithms SIMP to them.

SUP and INF.

SUP and INF are each called with two arguments, J and L. J is an expression and L Is a list. SUP(INF) attempts to find the largest (smallest) value that J can have consistent with the Inequalities in S. (See definitions of $\sup_S x_k$ and $\inf_S x_K$ in Section 2).

L is a set of variables. The first call to SUP (or to INF) is usually given with NIL for L; variables are then sometimes added to L by recursive calls to SUP and INF. See Tables I, II.

SUPP and INFF.

SUPP and INFF are called with arguments x and y. x is a variable and y is an expression. SUPP is called by SUP when J is a variable and J *4* L. Similarly INFF is called by INF. SUPP is designed to handle the case when SUP(J,L) returns an answer which contains J itself. See Tables III, IV.

The action of SUP can be viewed as putting together a string of inequalities. For example, If S consists of the inequalities **$[(0 \leq J \leq k) \wedge (0 \leq k \leq 3)]$** then SUP(J,NIL) will determine **$(J \leq k \leq 3)$,** and return 3 as the correct value. **However, if S consists of $[(0 \leq J \leq k) \wedge (0 \leq k \leq 6\text{-}J)]$** then gets **$(J \leq k \leq 6\text{-}J)$** and It must solve this inequality to determine: **$(2J \leq 6)$, $(J \leq 3)$,** and again return the correct value 3. This type of "solving" is done by the algorithm SUPP. That is, SUPP (and analogously INFF) handles the cases when SUP(J,L) might return an expression containing J itself.

For instance, In the above example, where S consists of **$[(0 \leq J \leq k) \wedge (0 \leq k \leq 6\text{-}J)]$,** since J is a variable and **J ∉ NIL,** the algorithm SUP (Step 2.2) finds the member **$(0 \leq J \leq k)$** of S with J as its middle term, and then puts Z: = (SUP k {J}). Since SUP(k,(J}) (eventually) returns the value (6-J), a call is made to SUPP(J, (6-J)) which returns the correct value 3. In evaluating SUPP (J, (6-J)), SUPP expresses (6-J) in the form b J +c, with b = - 1, c = 6, and returns the

value $\frac{c}{1-b} = \frac{6}{1+1} = 3.$

On the other hand if S had consisted of $[(0 \leq J \leq k) \wedge (0 \leq k \leq 6+J)]$, then SUPP would have been called with arguments J and (6+J) which would lead to b = 1, c = 6, and result in the correct value $+\infty$ for SUP(J,NIL). That this is the correct answer can be seen as follows: From S we get $(J \leq k \leq 6+J)$, which implies $0 \leq 6$. But this is no restriction at all on J, and hence $\sup_S J = +\infty$.

Examples.

In Section 5 of [8], Tables I-IV and the sup-inf method, are used to prove (or disprove) Examples 1-4 below, and to compute the indicated values of SUP and INF given in Examples 5-9 below.

1. $(x < 1 \rightarrow x < 2)$

2. $(5x < 11 \rightarrow 7x < 16)$

3. $(5x < 16 \rightarrow 7x < 16)$ **FALSE**

4. $(2y + 3 \leq 5z \wedge z \leq x - y \wedge 3x \leq 5 \rightarrow 2y \leq 3)$

5. S = ([j: 0 k][k: 0 j+L][L: 0 2-j])
$\sup_S$ k = 2 = SUP(k,NIL)

6. S = ([j: 0 L][k: j j][L: 0 (2j - 3k)])
not "natural" (see Section 2)
$\inf_S$ j = $\inf_S$ k = $\inf_S$ L = 0 = $\sup_S$ j = $\sup_S$ k = $\sup_S$ k ,
INF(J,NIL) = INF(k,NIL) = INF(L,NIL) = 0,
SUP(j,NIL) = SUP(L,NIL) = 0, SUP(k,NIL) = $+\infty$.

7. $S = ([j: \max(j, \frac{L}{2} + \frac{3}{2}k) \min(k,L)]$

$[k: j \min(j, \frac{2}{3}j - \frac{L}{3})][L: j (2j - 3k)])$

(This is the naturalized form of 6.)
SUP(k,NIL) = 0. Other values of SUP and INF are 0 as in 6.

8. S = ([k: 0 j][j: 0 L][L: 0 j-k+4]).
$\sup_S$ k = 4, SUP(k,NIL) = $+\infty$.
When S is naturalized, SUP(k,NIL) = 4.

9. S = ([J: 0 k+L][k: 0 5][L: 0 k]).
INF(j,NIL) = INF(k,NIL) = INF(L,NIL) = 0,
SUP(j,NIL) = 10, SUP(k,NIL) = SUP(L,NIL) = 5.

Acknowledgments. I am indebted to Mabry Tyson, Michael Ballantyne and Mark Moriconi for their help on this paper.

Table I

ALGORITHM SUP(J,L)

	IF	ACTION	RETURN
1.	J is a number		J
2.	J is a variable		
2.1	J ∈ L		J
2.2	J ∉ L	Put b: = UPPER(S,J) Put Z: = SUP(b,L ∪ {J})	SUPP(J,Z)
3.	J = ("-"A)		("-"INF(A,L))
4.	J = "/"A)		("/"INF(A,L))
5.	J = ("*"A N), where N is a number		
	N > 0		("*"SUP(A,L)N))
	N < 0		("*"INF(A,L)N))
	N = 0		0
6.	J = ("+"A B), where A has the form ("*"r A'), where r is a number and A' is a variable	Put B': = SUP(B,L ∪ {A'})	
6.1	B' = B		("+"SUP(A,L)B')
6.2	B' ≠ B	Put J': = SIMP("+"A B')	
6.2.1	J' = ("+"A B')		("+"SUP(A,L)B')
6.2.2	J' ≠ ("+"A B')		SUP(J',L)
7.	J = ("+"A B)	Put J': = SIMP("+"SUP(A,L)SUP(B,L))	
7.1	J' = J		+∞
7.2	J' ≠ J		SUP(J',L)
8.	J = ("max"A B)		MAX(SUP(A,L), SUP(B,L))
9.	J = ("min"A B)		MIN(SUP(A,L), SUP(B,L))
10.	Otherwise		+∞

Table II

ALGORITHM INF(J,L)

	IF	ACTION	RETURN
1.	J is a number		J
2.	J is a variable		
2.1	J ∈ L		J
2.2	J ∉ L	Put a: = LOWER(S,J) Put Z: = INF(a,L ∪ {J})	INFF(J,Z)
3.	J = ("-"SUP(A,L))		("-"SUP(A,L))
4.	J = ("/"A)		("/"SUP(A,L))
5.	J = ("*"A N)		
	N > 0		("*"INF(A,L)N)
	N < 0		("*"SUP(A,L)N)
	N = 0		0
6.	J = ("+"A B), where A has the form ("*"r A'), where r is a number and A' is a variable	Put B': = INF(B,L ∪ {A'})	
6.1	B' = B		("+"INF(A,L)B)
6.2		Put J': = SIMP("+"A B')	
6.2.1	J' = ("-"A B')		("+"INF(A,L)B')
6.2.2	J' ≠ ("+"A B')		INF(J',L)
7.	J = ("+"A B)	Put J': = SIMP("+"INF(A,L)INF(B,L))	
7.1	J' = J		-∞
7.2	J' ≠ J		INF(J',L)
8.	J = ("max"A B)		MAX(INF(A,L), INF(B,L))
9.	J = ("min"A B)		MIN(INF(A,L), INF(B,L))
10.	Otherwise		-∞

Table III

ALGORITHM SUPP(x,y)

	IF	ACTION	RETURN
1.	y is a number		y
2.	x = y		$+\infty$
3.	x is not a variable		$+\infty$
4.	y = ("max"A B)		MAX(SUPP(x,A), SUPP(x,B))
5.	y = ("min"A B)		MIN(SUPP(x,A), SUPP(x,B))
6.	"min" or "max" occurs in y	Pull "min" or "max" to fromt of y[8], getting y'.	SUPP(x,y')
7.	Otherwise express y as b x + c, where x does not occur in b or c.		
7.1	b = 0		y
7.2	b not a number		$+\infty$
7.3	b < 1		$\frac{c}{1-b}$
7.4	1 < b		$+\infty$
7.5	b = 1		
7.5.1	c is not a number		$+\infty$
7.5.2	c < 0		$-\infty$
7.5.3	c $\geq$ 0		$+\infty$

Table IV

ALGORITHM INFF(x,y)

	IF	ACTION	RETURN
1.	y is a number		y
2.	x = y		0
3.	x is not a variable		$-\infty$
4.	y = ("max"A B)		MAX(INFF(X,A), INFF(x,B))
5.	y = ("min"A B)		MIN(INFF(X,A), INFF(x,B))
6.	"min" or "max" occurs in y	Pull "min" or "max" to front of y[8], getting y'	INFF(x,y')
7.	Otherwise express y as b x + c, where x does not occur in b or c.		
7.1	b = 0		y
7.2	b not a number		0
7.3	b < 1		$\frac{c}{1-b}$
7.4	1 < b		0
7.5	b = 1		
7.5.1	c is not a number		0
7.5.2	c > 0		$+\infty$
7.5.3	c $\leq$ 0		0

It is clear that the methods of integer programming [11] can be used to prove theorems of the type we consider here, and similarly our procedure can be thought of as another method for solving integer programming problems.

[8] For example, $y_1 + (\text{"max"} y_2 y_3)$ is converted to $(\text{"max"} (y_1 + y_2)(y_1 + y_3))$.

References

1. Cooper, D.C. Programs for mechanical program verification. Mach. Intell. 6. American Elsevier, New York, 1971. 43-59.

2. Davis, M. A program for Presburger'a algorithm. Summer Inst, for Symbolic Logic. Cornell U., 1957. 215-233.

3. Presburger, M. Uber die Vollstandlgkeit eines genissen Systems der Arithmetlk ganzer Axhlen, in Welshem die Addition als einzige Operation hervortritt, Sprawozdanie z I Kongresu Matetnatykow Krajow Slowcanskich Warszawa, 1929, pp. 92-101.

4. Wang, Hao. Toward Mechanical Mathematics, IBM J. Res. Dey. 4, 2-22.

5. Bledsoe, W.W., Boyer, R.S. and Henneman, W.H. Computer proofs of limit theorems. Artif. Intell. 3, 1972. 27-60.

6. Good, D.I., London, R.L., Bledsoe, W.W. An Interactive Verification System. Proceedings of the 1975 International Conf. on Reliable Software, Los Angeles, April, 1975, pp. 482-492, and IEEE Trans, on Software Engineering, 1(1975), 59-67.

7. Bledsoe, W.W. and Tyson, M. Typing and proofs by cases in program verification. The University of Texas at Austin Mathematics Department Memo ATP 15, May 1975.

8. Bledsoe, W.W. The Sup-Inf Method in Presburger Arithmetic, The University of Texas at Austin Mathematics Department Memo ATP 18, December, 1974. (This Is essentially the same as the present paper except that It contains proofs of the theorems in Section 4.)

9. Bledsoe, W.W. and Tyson, Mabry. The UT Interactive Prover. The University of Texas at Austin Mathematics Department Memo ATP 17, May 1975.

10. Shostak, Robert. Private Communication, Stanford Research Inst., March 1975.

11. Gomery, R.E, An algorithm for Integer solutions to linear programs. Princeton-IBM Mathematics research project. Technical Report No. 1, 1958.