The Sup-Inf Method in Presburger Arithmetic

by

W. W. Bledsoe

December 1974 ATP-18

* This work was supported by NSF Grant #DCR74-12866.

The Sup-Inf Method in Presburger Arithmetic

by

W.W. Bledsoe

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.

This method has been programmed and used extensively as a part of an automatic theorem proving system.

The Sup-Inf Method in Presburger Arithemtic

by

W.W. Bledsoe

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 $\leq$ and $=$, the propositional 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:

(i) Elimination of quantifiers (and replacement of variables by constants)

(ii) Evaluation of the resulting formula (which has no variables) to determine its validity.

Cooper's method

Cooper [1] utilizes such a proceedure. For example his method would convert the theorem

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

successively to

$$\sim \exists x(x < 1 \wedge 1 < x) \quad ,$$

$$\sim \bigvee_{j=1}^{1} [1+j < 1 \wedge 1 < 1+j] \quad ,$$

$$\sim [2 < 1 \wedge 1 < 2] \quad ,$$

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 successively to

$$\sim \exists x(5x < 11 \wedge 15 < 7x)$$

$$\sim \exists x(35x < 77 \wedge 75 < 35x)$$

$$\sim \exists x(x < 77 \wedge 75 < x \wedge x \equiv 0 (\text{mod } 35))$$

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

which requires testing the expression for each of the integers, $j = 1, 2, \ldots, 35$. 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, which is described below, handles the case of Presburger formulas with only 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 [1,6,7,8,9,10,11]. There a theorem about the program, is required to be proved. Such theorems are often <u>not</u> 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:

$$\forall j\{j \leq 4 \wedge \forall k(k \leq 5 \longrightarrow A[k] \leq A[k+1]) \longrightarrow A[j] \leq A[j+1]\} \tag{1}$$

Backchaing on the second hypothesis generates the subgoal

(2) $$(j \leq 4 \rightarrow j \leq 5)$$

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[1] in the proof of (2). 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. It is not clear whether the methods of this paper can be extended to the general case where both universal and existential quantifiers are present.

First let us note that formula (2) is easily verified by adding the negation of the conclusion to the hypothesis

$$(j \leq 4 \wedge 6 \leq j)$$

and combining to get the contradiction

$$(6 \leq j \leq 4) \quad .$$

Our proceedure does essentially the same thing on this example. We now describe Section 2 our procedure for determining the validity of universally quantified Presburger formulas.

---

[1]The skolemization process will not be discussed here. These universally quantified variables (or skolem constants) can be thought of as "arbitrary but fixed", whereas the existentially quantified variables (or "skolem variables") are not fixed but can be replaced by other expressions. See [4] and [5, p.37, footnote 12]. Since we will only be working with universally quantified variables in this paper we will call them "variables".

In Section 3 we define the pivotal algorithms SUP and INF and prove in Section 4 that they terminate with desirable outputs when applied to quasi-linear formulas.

Several examples are given in Section 5.

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

Notice that F has 3 (universally quantified) 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_i's. (Expressions (5'),(7'), and (8') correspond to expressions (5),(7), and (8) given later in our general procedure).

We first obtain

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

as the negation of F, and then convert it to

(7') $$\begin{aligned} &(x_3 + x_2 \leq x_1 < \infty) \wedge (0 \leq x_1 \leq \tfrac{5}{3}) \\ &\wedge (0 \leq x_2 \leq \tfrac{5}{2} x_3 - \tfrac{3}{2}) \wedge (0 \leq x_2 \leq x_1 - x_3) \wedge (2 \leq x_2 < \infty) \\ &\wedge (\tfrac{2}{5} x_2 + \tfrac{3}{5} \leq x_3 < \infty) \wedge (0 \leq x_3 \leq x_1 - x_2) \quad , \end{aligned}$$

and finally to

(8') $$\begin{aligned} &(x_3 + x_2 \leq x_1 \leq \tfrac{5}{3}) \\ \wedge \quad &(2 \leq x_2 \leq \min(\tfrac{5}{2} x_3 - \tfrac{3}{2} \, , \, x_1 - x_3)) \\ \wedge \quad &(\tfrac{2}{5} x_2 + \tfrac{3}{5} \leq x_3 \leq x_1 - x_2) \end{aligned}$$

---

[2] How $3 \leq 2x_2 - 1$ is derived from $2x_2 \not\leq 3$ is explained in Section 3.

To show the invalidity of (8') we calculate a lower bound $\underline{x}_1$ for x_1 and an upper bound $\overline{x}_1$, and check that the interval $[\underline{x}_1, \overline{x}_1]$ contains no integer. For this example, $\underline{x}_1 = \frac{17}{5}$ and $\overline{x}_1 = \frac{5}{3}$, and since $\frac{5}{3} < \frac{17}{5}$ we are finished. $\underline{x}_1$ and $\overline{x}_1$ are computed by the algorithms INF and SUP given in Section 3 (See Example 4, Section 5).

Quasi-Linear Formulas.

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. Of course Presburger formulas with only universally quantified variables are special cases of quasi-linear formulas.

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

$$(a_1 \leq x_1 \leq b_1) \wedge (a_2 \leq x_2 \leq b_2) \wedge \ldots \wedge (a_n \leq x_n \leq b_n) \quad ,$$

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:

<u>The Sup-Inf Procedure</u>.

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

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

and the A_i, B_i, C_i, D_i are quasi-linear expressions in $x_1, x_2, \ldots, x_n$. We

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

eliminate the equalities in (4) by converting each $(C_i = D_i)$ into $(C_i \leq D_i \wedge D_i \leq C_i)$[4] so that each G_i has the form

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

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

$$(6) \qquad (a_{i1} \leq x_1 \leq b_{i1}) \wedge (a_{i2} \leq x_2 \leq b_{i2}) \wedge \ldots \wedge (a_{in} \leq x_n \leq b_{in})$$

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 $a_{ij} = 0$, $b_{ij} = \infty$.

So (5) is converted to

$$(7) \quad \bigwedge_{i=1}^{m} (a_{i1} \leq x_1 \leq b_{i1}) \wedge \bigwedge_{i=1}^{m} (a_{i2} \leq x_2 \leq b_{i2}) \wedge \ldots \wedge \bigwedge_{i=1}^{m} (a_{in} \leq x_n \leq b_{in})$$

and finally (7) is converted to

$$(8) \qquad (a_1 \leq x_1 \leq b_1) \wedge (a_2 \leq x_2 \leq b_2) \wedge \ldots \wedge (a_n \leq x_n \leq b_n) \quad ,$$

---

[4] In practice we do not always convert the equalities to inequalities, but rather use a "substitution of equals" technique to gain efficiency. See [13,14].

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

where, for $k = 1, n$,

$$a_k = (\max a_{1k}\ a_{2k}\ \ldots\ a_{mk})^6$$
$$b_k = (\min b_{1k}\ b_{2k}\ \ldots\ b_{mk}) \quad .$$

Thus, by this whole process we convert $\sim F$ to a disjunct

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

where each F_L has the form

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

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

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

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

contains no integer,

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

for some k, $k = 1, 2, \ldots, n$, where $\sup_S x_k$ and $\inf_S x_k$ are defined as follows:

DEFINITIONS.

If S is a set of inequalities of the form (9) and $x_1', x_2', \ldots, x_n'$ are real numbers, then $(x_1', x_2', \ldots, x_n')$ 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 $x_1', x_2', \ldots, x_n'$ 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 $x_1, x_2, \ldots, x_n$, then $\sup_S A$ is defined to be the least upper bound of all numbers

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

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 has been reduced to determining $\sup_S x_k$ and $\inf_S x_k$, 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 we prove that if S is any set of inequalities of the form (9) where we assume only that the a_{Li}, b_{Li} are quasi-linear in $x_1, \ldots, x_n$, then the outputs SUP(x_k, NIL) and INF(x_k, NIL) have the property

$$\mathrm{INF}(x_k, \mathrm{NIL}) \leq \inf x_k , \quad \sup x_k \leq \mathrm{SUP}(x_k, \mathrm{NIL}) .$$

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.,

$$\mathrm{INF}(x_k, \mathrm{NIL}) = \inf x_k , \quad \sup x_k = \mathrm{SUP}(x_k, \mathrm{NIL}) .$$

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

$$[\mathrm{INF}(x_k, \mathrm{NIL}), \mathrm{SUP}(x_k, \mathrm{NIL})]$$

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

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 King in [6]. 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

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

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

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

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

$$(11) \qquad (x_3 \le 5x_2 \longrightarrow x_3 \le 8) \ .$$

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

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

which is used to prove $x_3 \le 8$.

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 method, which is our desire to efficiently handle certain "proof by cases". This is best illustrated by an example taken from [12]. Suppose we are to prove the theorem

(12) $$(K \leq 3 \rightarrow K \leq 1 \ \vee \ 2 \leq K \leq 3)$$

where K is a variable. The negation of (12) is converted successively to

$$(K \leq 3 \ \wedge \ K \not\leq 1 \ \wedge \ (2 \not\leq K \ \vee \ K \not\leq 3)) \ ,$$

$$(K \leq 3 \ \wedge \ 2 \leq K \ \wedge \ K \leq 1) \ \vee \ (K \leq 3 \ \wedge \ 2 \leq K \ \wedge \ 4 \leq K) \ ,$$

$$(2 \leq K \leq 1) \ \vee \ (4 \leq K \leq 3) \ ,$$

for which a contradiction is easily reached.

However, suppose (12) was presented in an equivalent form

(13) $$(K \leq 3 \wedge (K \leq 1 \rightarrow C) \wedge (2 \leq K \leq 3 \rightarrow C) \rightarrow C)$$

which is <u>not</u> a formula in Presburger arithmetic. If we back chain off of the second hypothesis, we obtain the subgoal

$$(K \leq 3 \rightarrow K \leq 1)$$

which is false. Similarly, if we backchain off of the third hypothesis we fail again.

It is difficult to imagine how an automatic procedure would start on (13). It could of course, be made to backchain on <u>both</u> the second and third hypothesis and thereby set up the subgoal (12), but this would be an unnatural preliminary activity. What is more, formula (13) is an abstraction from formula (14) below, which is part of a verification condition which appeared in the proof of a sort program (see King [6]).

$$
\begin{array}{lll}
 & & ((1 \leq N) \\
 & \wedge & \forall m(2 \leq N \wedge 1 \leq m \wedge m \leq 1 \longrightarrow A[m] \leq A[2]) \\
(14) & \wedge & \forall k(k + 1 \leq N \wedge 2 \leq k \longrightarrow A[k] \leq A[k+1]) \\
 & \longrightarrow & K(K + 1 \leq N \wedge 1 \leq K \longrightarrow A[K] \leq A[K+1]))
\end{array}
$$

It is even less clear how an automatic procedure would proceed to set up a solvable Presburger problem from (14).

The procedure we employ to prove theorems like (13) (and similarly for (14)), stores the hypothesis $K \leq 3$ as

$$(15) \qquad (0 \leq K \leq 3)$$

and proceeds to prove

$$(16) \qquad ((K \leq 1 \longrightarrow C) \wedge (2 \leq K \leq 3 \longrightarrow C) \longrightarrow C) \quad .$$

By backchaining off of the first hypothesis of (16) it obtains the subgoal

(17) $(K \leq 1)$

which is supposed to be proved from (15) but which cannot be done. However in <u>trying</u> to prove (17) from (15), i.e.,

(18) $(0 \leq K \leq 3) \longrightarrow K \leq 1)$,

it updates (15) with the negation of (17) getting

$$(0 \leq K \leq 3) \wedge (K \nleq 1)$$

or

(15') $(2 \leq k \leq 3)$.

Now (15') does not represent a contradiction and hence we have not established (17) from (15). However we have shown that <u>except for the case</u> $(2 \leq k \leq 3)$ we <u>have</u> proved (17). That is to say, only the case $(2 \leq k \leq 3)$ needs to be further considered in establishing our original goal (16). So we take (15') as an additional new hypothesis, and try again to prove (16).

Now backchaining off of the second hypothesis of (16) we obtain the subgoal $(2 \leq k \leq 3)$ which follows immediately from (15'), and the proof is complete.

This technique <u>forced</u> the prover to consider the two cases $(K \leq 1)$ and $(2 \leq K \leq 3)$. Once these cases were treated separately the proof went easily.

Notice that in this example we did not immediately put theorem (13) in disjunctive normal form. But rather did so in (18) after we had obtained a Presburger formula.

The program which carries out this procedure is part of the interactive prover described in [13]. In that program the set S of inequalities from (9) are carried in a special hypothesis called TYPELIST. TYPELIST is simply a set of triples

$$(\{x_1: a_{L1}\ b_{L1}\}\{x_2: a_{L2}\ b_{L2}\}\ldots\{x_n: a_{Ln}\ b_{Ln}\})$$

Each x_i is said to be "typed": it has type, non-negative integer, and, furthermore, its interval restriction

$$a_{Li} \leq x_i \leq b_{Li}$$

is thought of as "interval typing". All such interval information (in the hypothesis) is carried in TYPELIST. When and if TYPELIST obtains a contradiction the proof is complete. Also whenever the prover is trying to prove another inequality, its negation is updated to TYPELIST and either

(1) a contradiction is found terminating the proof

or

(2) this updated TYPELIST is passed back as "cases" information.

The reader can consult [13] for details of this process.

3. Algorithms

Here we describe the pivotol algorithms SUP and INF, and a few others that support them.

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., $\frac{1}{r}*A$.

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 $\sup x_j$ and $\inf x_j$ for a given S. S will be assumed to be fixed throughout the remainder of this section, and in Section 4.

Before we give the algorithms we list some preliminary conventions and definitions.

<u>Convention</u>. By <u>x is a number</u> we will mean (here) that x is a rational number or ∞ or $-\infty$. (We will assume that $-\infty \leq x \leq \infty$ for all numbers x).

Definition.

$$\text{MAX}(x,y) = \begin{cases} y & \text{if } x \leq y \\ x & \text{if } y < x \\ (\text{"max" } x\ y) & \text{if } x \text{ and } y \text{ are not both numbers.} \end{cases}$$ [7]

Similarly for MIN(x,y).

Let $V = \{x_1, x_2, \ldots, x_n\}$ be the set of all variables (universally quantified variables, see Section 2) occurring in the given set S of inequalities.

If J is in V 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

[7] In case x and y are not both numbers we want the result to be the triple whose first member is the symbol "max" and whose second and third members are x and y. Thus MAX(2,5) is 5, whereas MAX(2,Z+1) is ("max" "2" "Z+1"). The symbols "min", "+", "-", etc., are handled similarly.

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 end of this section for more details on it. 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 (i.e., a subset of V). The first call to SUP (or to INF) is usually given with NIL for L; members of V are then sometimes added to L by recursive calls to SUP and INF.

ALGORITHM SUP(J,L)

IF	ACTION	RETURN
1. J is a number		J
2. J is a variable		
2.1 $J \in L$		J
2.2 $J \notin L$	Put b: = UPPER(S,J)[8]	
	Put Z: = SUP(b, $L \cup \{J\}$)	SUPP(J,Z)
3. J = ("-"A)[9]		("-"INF(A,L))
4. J = ("/"A)		("/"INF(A,L))
5. J = ("*"AN), where N is a number		
$N > 0$		("*"SUP(A,L)N))
$N < 0$		("*"INF(A,L)N))
$N = 0$		0
6. J = ("+"A B),[10] where A has the form ("*"r A'), where r is a number and A' is a variable		
	Put B': = SUP(B, $L \cup \{A'\}$)	
6.1 B' = B		("+" SUP(A,L)B')
6.2 B' $\neq$ B	Put J': = SIMP("+" A B')	
6.2.1 J' = ("+" A B')		("+" SUP(A,L)B')
6.2.2 J' $\neq$ ("+" A B')		SUP(J',L)

---

[8] As explained earlier, if $(a \leq J \leq b)$ is in S, then UPPER(S,J) returns b, and LOWER(S,J) returns a.

[9] If J is an expression whose first member is the symbol "-", and second member is a formula A, then the algorithm returns ("-"J') where J' is the result of a call to INF(A,L). "-" is the minus operator and "/" is the inverse operator.

[10] Step 6 could be omitted and we would still retain the desired property $J \underset{S}{\leq} SUP(J,L)$ (See Theorem 2). However if Step 6 is omitted SUP(J,NIL) does not always give the best bound, $\sup_S J$. See example 5, Section 5.

7. J = ("+" A B)	Put J': = SIMP("+"SUP(A,L)SUP(B,L))	
7.1 J' = J		$+\infty$
7.2 J' $\neq$ J		SUP(J',L)
8. J = ("max"A B)[11]		MAX(SUP(A,L),SUP(B,L))
9. J = ("min"A B)		MIN(SUP(A,L),SUP(B,L))
10. Otherwise		$+\infty$

[11] We mean here that y is a triple whose first term is the symbol "max" and whose second and third terms are formulas which we will call A and B.

ALGORITHM INF(J,L)

IF	ACTION	RETURN
1. J is a number		J
2. J is a variable		
2.1 $J \in L$		J
2.2 $J \notin L$	Put a: = LOWER(S,J)[8]	
	Put Z: = INF(a,L $\cup$ {J})	INFF(J,Z)
3. J = ("-"A)		("-"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 $\cup$ {A'})	
6.1 B' = B		("+"INF(A,L)B)
6.2 B' $\neq$ B	Put J': = SIMP("+"A B')	
6.2.1 J' = ("+"A B')		("+"INF(A,L)B')
6.2.2 J' $\neq$ ("+"A B')		INF(J',L)
7. J = ("+"A B)	Put J': = SIMP("+"INF(A,L)INF(B,L))	
7.1 J' = J		$-\infty$
7.2 J' $\neq$ 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		$-\infty$

SUPP and INFF.

SUPP and INFF are called with arguments x and y. x is a variable (a member of V) and y is an expression. SUPP is called by SUP when $J \in V$ and $J \notin L$. Similarly INFF is called by INF. SUPP is designed to handle the case when SUP(J,L) returns an answer which contains J itself. Further explanation and examples are given immediately after the statement of the algorithms.

ALGORITHM SUPP(x,y).

IF	ACTION	RETURN
1. y is a number		y
2. x = y		$+\infty$
3. $x \notin V$		$+\infty$
4. y = ("max"A B)[11]		MAX(SUPP(x,A),SUPP(x,B))
5. y = ("min"A B)		MIN(" , ")
6. "min" or "max" occurs in y	Pull "min" or "max" to front of y[12], 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 $c \geq 0$		$+\infty$

[12] For example, y_1 + ("max"$y_2 y_3$) is converted to ("max"$(y_1+y_2)(y_1+y_3)$).

ALGORITHM INFF(x,y).

IF	ACTION	RETURN
1. y is a number		y
2. x = y		0
3. $x \notin V$		$-\infty$
4. y = ("max"A B)		MAX(INFF(X,A),INFF(x,B))
5. y = ("min"A B)		MIN(" , ")
6. "min" or "max" occurs in y	Pull "min" or "max" to front of y[12], 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

The action of SUP can be viewed as putting together a string of inequalities. For example, if S consist of the inequalities

$$(0 \leq J \leq k) \ (0 \leq k \leq 3)$$

then SUP(J,NIL) will determine

$$J \leq k \leq 3 \ ,$$

and return 3 as the correct value. However, if S consists of

$$(0 \leq J \leq k) \ (0 \leq k \leq 6\text{-}J)$$

then it gets

$$J \leq k \leq 6\text{-}J$$

and it must solve this inequality to determine

$$2J \leq 6 \ , \qquad J \leq 3$$

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

$$(0 \leq J \leq k)\ (0 \leq k \leq 6\text{-}J) \quad ,$$

since J is a variable and $J \notin$ 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

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

On the other hand if S had consisted of

$$(0 \leq J \leq k)\ (0 \leq k \leq 6+J) \quad ,$$

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

$$J \leq k \leq 6+J \quad ,$$

which implies

$$0 \leq 6 \quad .$$