But this is no restriction at all on J, and hence $\sup_S J = +\infty$.

SUP(J,L) and INF(J,L) always return (quasi-linear) expressions in L. This is proved in Section 4. In case J contains a symbol F which is not a variable and is not one of the connectives, +, -, *, max, min, /, then SUP and INF, (last step), return the value $+\infty$ or $-\infty$ for it. Thus all such "foreign objects" are removed from the expressions returned by SUP(J,L) and INF(J,L).

We also show in Section 4 that SUP(J,NIL) and INF(J,NIL) always return numbers or $\pm\infty$.

Converting $<$ to $\leq$.

If x and y are variables then the expression $(x < 3y+5)$ is automatically converted to $(x \leq 3y+4)$, because, as variables, x and y represent non-negative integers. Since we allow rational number coefficients in our quasi-linear expressions, these also must be reckoned with. For example, the expression $(\frac{2}{3}x < \frac{4}{5}y + 5)$ is converted first to $(10x < 12y+75)$ and then to $(10x \leq 12y + 74)$. In general $(A < B)$ is automatically converted to $(gA \leq gB-1)$ where g is the greatest common denominator of the (rational) coefficients in the quasi-linear expressions A and B. Similarly, $\sim(A \leq B)$ is automatically converted to $(gB \leq gA-1)$.

Simplification

SIMP is a simplification routine which drives expressions to a canonical form. These are well known in the literature [15,5,8,etc.] and will not be belabored here. If A is a linear expression in the variables $x_1,\ldots,x_n$, then SIMP(A) is an equivalent expression

$$(x_1' \; r_1 + (x_2' \; r_2 + \ldots + (x_n' \; r_n + r_{n+1})\cdots))$$

where $x_1',\ldots,x_n'$ is a permutation of $x_1,\ldots,x_n$, and $r_1, r_2,\ldots,r_{n+1}$ are rational numbers. If one of the r_i is 0 then that term is omitted. The order of the x_i' is fixed at each call to SIMP in our program, but it could just as well be fixed once for all such calls.

Does [a,b] Contain an Integer.

In the sup-inf procedure we must test whether the interval $[\mathrm{INF}(x_i,\mathrm{NIL}), \mathrm{SUP}(x_i,\mathrm{NIL})]$ contains an integer. We do this with the algorithm INT(a,b) defined as follows. Each of a and b is either an integer or $+\infty$ or $-\infty$ or a rational numbers (expressed as ("*" p ("/" q)), where p and q are integers).

IF	RETURN
$a = +\infty$ or $b = -\infty$	FALSE
$a = -\infty$ or $a = +\infty$	TRUE
$b < a$	FALSE
$b \geq a + 1$	TRUE
$a = b$ and a is an integer	TRUE
$a \neq b$ and Integerpart (a) < Integerpart (b)*	TRUE
Else	FALSE

If x is a rational number expressed in the form ("" p("/"q)), then Integerpart (x) is gotten in LISP by dividing p by q, (DIV p q).

4. Properties of SUP and INF

The algorithms SUP and INF, defined in Section 3, play a crucial role in the sup-inf procedure. They attempt to compute the values $\sup_S x$ and $\inf_S x$ as defined on page 9 of Section 2, and must possess the properties

$$\text{INF}(x,\text{NIL}) \leq \inf_S x \ , \quad \sup_S x \leq \text{SUP}(x,\text{NIL})$$

which we prove as Theorem 5 below.

The set S is fixed throughout this section (see Sections 2 and 3 definitions).

In the proofs given below we will employ a recursive proof principle[13] which can be stated as follows:

<u>Recursive Proof Principle</u>.

Let $\mathcal{F}$ be an algorithm which accepts a vector x as an input and returns a value f(x) whenever it terminates. If we wish to prove the theorem

(1) $$P(x) \longrightarrow Q(f(x)) \ ,$$

[13] This is just a special case of Burstall's structural induction [16, p.42], and of McCarthy's recursive induction [17].

and it is given that $\mathcal{F}$ terminates with a value $f(x)$ for each x for which $P(x)$ holds, then it suffices to prove (1) under the assumption that

(2) $$P(y) \longrightarrow Q(f(y))$$

holds for each recursive call to $\mathcal{F}$ from $\mathcal{F}$ in the computation of $f(x)$.

In the proofs that follow the assumption (2) will be referred to as an "induction hypothesis", and proofs are said to be done by "recursive induction".

When more than one algorithm is involved in such a proof, the principle is extended appropriately. For example, in the case of two algorithms $\mathcal{F}$ and $\mathcal{G}$, if we are trying to prove

(1') $$(P(x) \longrightarrow Q(\mathcal{F}(x))) \wedge (P'(x) \longrightarrow Q'(\mathcal{G}(x))) \quad ,$$

it suffices to prove (1') under the assumption that

(2') $$(P(y) \longrightarrow Q(\mathcal{F}(y)))$$

holds for each recursive call of $\mathcal{F}$ from $\mathcal{F}$ or from $\mathcal{G}$, and that

(2") $$(P'(y) \longrightarrow Q'(\mathcal{G}(y)))$$

holds for each recursive call of $\mathcal{G}$ from $\mathcal{F}$ or from $\mathcal{G}$. In all cases termination must be proved separately.

Let $V = \{x_1, x_2, \ldots, x_n)$, the variables of S, and let S be a set of inequalities of the form

$$a \leq x_j \leq b \quad ,$$

where a and b are quasi-linear in V. A number vector $\overline{x}' = (x_1', x_2', \ldots, x_n')$ is said to satisfy S if all of its inequalities hold when the x_i are replaced by the corresponding numbers x_i'.

Let E be a set of such $\overline{x}'$ which satisfy S. In the following A, B, C, D, etc., are quasi-linear expressions in V. Let $\overline{x}$ denote the vector $\overline{x} = (x_1, x_2, \ldots, x_n)$.

<u>Definition</u>. $A \underset{E}{\leq} B$ means that

$$A(\overline{x}'/\overline{x}) \leq B(\overline{x}'/\overline{x})$$

for every $\overline{x}' \in E$. That is, if $(x_1', x_2', \ldots, x_n')$ is in E, and A' is gotten from A by replacing each x_i by the number x_i', and similarly for B', then

$$A' \leq B' \quad .$$

We define $A \underset{E}{\equiv} B$, and $A \underset{E}{\geq} B$ in an analogous manner

The predicates $\underset{E}{\leq}$ and $\underset{E}{\equiv}$ act very much like $\leq$ and $=$. Some of their properties are given in Lemmas 1 - 3, and Theorem 4 which are given without proof.

Lemma 1.

1. $(A = B \longrightarrow A \underset{E}{\equiv} B)$

2. $(A \leq B \longrightarrow A \underset{E}{\leq} B)$

3. $(A \underset{E}{\leq} B \wedge B \underset{E}{\leq} C \longrightarrow A \underset{E}{\leq} C)$

4. $E \neq \emptyset \wedge A \underset{E}{\leq} B \wedge C \underset{E}{\leq} D \longrightarrow$

$$A + C \underset{E}{\leq} B + D \wedge (\text{"max"}\ A\ C) \underset{E}{\leq} (\text{"max"}\ B\ D)$$

5. $A \underset{E}{\leq} B \longrightarrow (\text{"-"}B) \underset{E}{\leq} (\text{"-"}A)$

6. $A \underset{E}{\leq} B \wedge N > 0 \wedge M < 0 \longrightarrow$

$$(\text{"*"}\ A\ N) \underset{E}{\leq} (\text{"*"}\ B\ N) \wedge (\text{"*"}\ B\ M) \underset{E}{\leq} (\text{"*"}\ A\ M)$$

Lemma 2.

1. ("*"A 0) $\underset{E}{\equiv}$ 0

2. SIMP(A) $\underset{E}{\equiv}$ A

SIMP is the simplification algorithm described in Section 3.

3. MAX(A,B) $\underset{E}{\equiv}$ ("max"A B)
The algorithms MAX and MIN are given in Section 3.
4. MIN(A,B) $\underset{E}{\equiv}$ ("min"A B)

Lemma 3. If $(a \leq J \leq b)$ is member of S and E is the set of $\bar{x}'$ satisfying S, then $a \underset{E}{\leq} J \underset{E}{\leq} b$.

Theorem 4. If E is the set of $\bar{x}'$ satisfying S, and $A \underset{E}{\leq} B$, then

$$\sup_S A \leq \sup_S B$$

and

$$\inf_S A \leq \inf_S B \quad .$$

Theorem 5. If $V = \{x_1, x_2, \ldots, x_n\}$, $J \in V$, S is a set of inequalities of the form

$$a \leq x_j \leq b \quad ,$$

where a and b are quasi-linear in V , then INF(J,NIL) and SUP(J,NIL) are numbers, and

$$\text{INF}(J,\text{NIL}) \leq \inf_S J \leq \sup_S J \leq \text{SUP}(J,\text{NIL}) \quad .$$

Proof. Let E be the set of number vectors $\bar{x}'$ which satisfy S.

By Theorem 6 below

(3) $$\text{INF}(J,\text{NIL}) \underset{E}{\leq} J \underset{E}{\leq} \text{SUP}(J,\text{NIL})$$

and also that INF(J,NIL) and SUP(J,NIL) are quasi-linear in NIL, and therefore are numbers (since NIL is the empty set). $\inf_S(\text{INF}(J,\text{NIL})) = \text{INF}(J,\text{NIL})$. $\text{Sup}_S(\text{SUP}(J,\text{NIL})) = \text{SUP}_S(J,\text{NIL})$, and by Theorem 4,

$$\text{INF}(J,\text{NIL}) \leq \inf_S J \leq \sup_S J \leq \text{SUP}(J,\text{NIL}) \quad .$$

Theorem 6. If J is quasi-linear in V and $L \subseteq V$, then

SUP(J,L) and INF(J,L) are quasi-linear in L ,

and

$$\text{INF}(J,L) \underset{E}{\leq} J \underset{E}{\leq} \text{SUP}(J,L) \quad ,$$

where E is the set of number vectors $\bar{x}'$ which satisfy S.

Proof. The proof is by recursive induction. Termination for SUP and INF is proved in Theorem 9 below.

The proof for INF is similar to that for SUP, which is given here by examining each step of the algorithm SUP.

The desired conclusion clearly holds for Steps 1,2.1,7.1, and 10. We now complete the proof for the other Steps of SUP.

Step 2.2. J is a variable but $J \notin L$.

In this case

(4) $$SUP(J,L) = SUPP(J,Z) \quad ,$$

where $Z = SUP(b, L \cup \{J\})$, $b = UPPER(S,J)$.

Since b is quasi-linear in V it follows by the induction hypothesis that

$$Z \text{ is quasi-linear in } L \cup \{J\}$$

and

(5) $$b \underset{E}{\leq} Z \quad ,$$

and by Theorem 7 below, that SUPP(J,Z) is quasi-linear in L.

But also

(6) $$(a \leq J \leq b)$$

is a member of S, (where $a = \text{LOWER}(S,J)$), so that by (6) and Lemma 3,

$$J \underset{E}{\leq} b$$

and hence by (5), $J \underset{E}{\leq} b \underset{E}{\leq} Z$, and by Theorem 8 below and (4) we have

$$\begin{aligned} J &\underset{E}{\leq} \text{SUPP}(J,Z) \\ &= \text{SUP}(J,L) \end{aligned}$$

as required.

<u>Step 3</u>. J has the form ("-"A).

In this case SUP(J,L) = ("-"INF(A,L)).

Thus by the induction hypothesis

INF(A,L) is quasi-linear in L ,

and

$$\text{INF}(A,L) \underset{E}{\leq} A \underset{E}{\leq} \text{SUP}(A,L) \quad .$$

Hence

("-"INF(A,L)) is quasi-linear in L ,

and using Lemma 1.5,

$$J = ("-"A) \underset{E}{\leq} ("-"INF(A,L)) = SUP(J,L) \quad .$$

Step 4. J has the form ("/"A).

This case does not arise except for positive rational numbers A, which cases are handled by Step 1.

Step 5. J has the form ("*"A N), where N is a number.

If $N = 0$, then by Lemma 2.1

$$J = ("*"A\ 0) \underset{E}{\equiv} 0 = SUP(J,L) \quad .$$

(Actually this case cannot arise since the simplifier, SIMP, would replace J by 0).

If $N > 0$, then using the induction hypothesis and Lemma 1.6,

$$\begin{aligned} J &= ("*"A\ N) \\ &\underset{E}{\leq} ("*"SUP(A,L)N) \\ &= SUP(J,L) \quad . \end{aligned}$$

If $N < 0$, then by the induction hypothesis

$$INF(A,J) \underset{E}{\leq} A \quad ,$$

and hence by Lemma 1.6,

$$\begin{aligned} J &= (\text{"*"}\ A\ N) \\ &\underset{E}{\leq} (\text{"*"}\,\mathrm{INF}(A,J)N) \\ &= \mathrm{SUP}(J,L)\quad . \end{aligned}$$

Steps 6 and 7. J has the form ("+" A B).

First by induction hypothesis $A \underset{E}{\leq} \mathrm{SUP}(A,L)$, $B \underset{E}{\leq} \mathrm{SUP}(B,L)$, $J' \underset{E}{\leq} \mathrm{SUP}(J',L)$ and $B \underset{E}{\leq} \mathrm{SUP}(B,\ L \cup \{A'\}) = B'$ and hence by Lemma 1.4,

$$J = (\text{"+"}\ A\ B) \underset{E}{\leq} (\text{"+"}\mathrm{SUP}(A,L)\quad \mathrm{SUP}(B,L))\quad ,$$

and

$$J = (\text{"+"}\ A\ B) \underset{E}{\leq} (\text{"+"}\mathrm{SUP}(A,L)\quad \mathrm{SUP}(B,\ L \cup \{A'\}))\ .$$

Thus using Lemmas 1.4 and 2.2 we obtain

$$\begin{aligned} J &\underset{E}{\leq} (\text{"+"}\mathrm{SUP}(A,L)\quad \mathrm{SUP}(B,\ L \cup \{A'\})) \\ &\underset{E}{\equiv} (\text{"+"}\mathrm{SUP}(A,L)\ \mathrm{SIMP}(B,\ L \cup \{A'\}))) \\ &= (\text{"+"}\mathrm{SUP}(A,L)B') = \mathrm{SUP}(J,L) \end{aligned}$$

which handles steps 6.1 and 6.2.1; and

$$\begin{aligned} J &= (\text{"+"}\ A\ B) \underset{E}{\leq} (\text{"+"}\ A\ B') \\ &\underset{E}{\equiv} \mathrm{SIMP}(\text{"+"}\ A\ B') = J' \underset{E}{\leq} \mathrm{SUP}(J',L) = \mathrm{SUP}(J,L) \end{aligned}$$

which handles step 6.2; and

$$\begin{aligned} J &\underset{E}{\leq} (\text{"+"SUP}(A,L)\ \text{SUP}(B,L)) \\ &\underset{E}{\equiv} \text{SIMP}(\text{"+"SUP}(A,L)\ \text{SUP}(B,L)) \\ &= J' \leq \text{SUP}(J',L) = \text{SUP}(J,L) \end{aligned}$$

which handles step 7.

Steps 8 and 9. J has the form ("max"A B) or ("min"A B).

$$J \underset{E}{\leq} \text{SUP}(J,L)$$

follows immediately from the induction hypothesis and Lemma 2.3, 2.4.

Theorem 7. If $x \in V$ and y is quasi-linear in L, then SUPP(x,y) is quasi-linear in L and does not contain x.

Proof. The proof is by recursive induction. Termination for SUPP is proved in Theorem 10 below.

The desired conclusion is obvious for all but Steps 4,5,6, and 7.3. It easily follows for Steps 4,5, and 6 by the induction hypothesis. (In Step 6, clearly y and y' contain the same members of V).

For Step 7 we have

$$y = b\,x + c$$

where b is a number, $b < 1$, and c does not contain x. Since by hypothesis, y is quasi-linear in L, it follows that $\frac{c}{1-b}$ is quasi-linear in L and does not contain x.

<u>Theorem 8</u>. If $x \in V$, y is quasi-linear in V, and $x \underset{E}{\leq} y$ then

$$x \underset{E}{\leq} \text{SUPP}(x,y) \quad .$$

<u>Proof</u>. The proof is by recursive induction. If y is a number, or if $\text{SUPP}(x,y) = y$ or if $\text{SUPP}(x,y) = +\infty$, then the conclusion clearly holds, so we need only consider Steps 4,5,6,7.3, and 7.5.2 of the SUPP algorithm.

<u>Step 4</u>. y has the form ("max" A B).

By hypothesis we have

$$x \underset{E}{\leq} (\text{"max" } A\ B) \quad .$$

In this case we cannot conclude that

(7) $$x \underset{E}{\leq} A$$

or that

$$x \underset{E}{\leq} B \quad .$$

However, if we define E_A to be the set of those members $\overline{x}'$ of E for which (7) holds, (i.e., (7) holds when the numbers x_i' are substituted for the x_i in x and A), and similarly define E_B, then we have

$$x \underset{E_A}{\leq} A \quad \text{and} \quad x \underset{E_B}{\leq} B \quad .$$

And hence by the induction hypothesis we have that

$$x \underset{E_A}{\leq} \text{SUPP}(x,A) \quad \text{and} \quad x \underset{E_B}{\leq} \text{SUPP}(x,B)$$

and hence, since $E = E_A \cup E_B$,

$$\begin{aligned} x &\underset{E}{\leq} (\text{"max" SUPP}(x,A)\ \text{SUPP}(x,B)) \\ &= \text{MAX}(\text{SUPP}(x,A),\ \text{SUPP}(x,B)) \\ &= \text{SUPP}(x,y) \quad . \end{aligned}$$

<u>Step 5</u>. y has the form ("min" A B).

By hypothesis we have

$$x \underset{E}{\leq} (\text{"min" } A\ B)$$

and hence

$$x \underset{E}{\leq} A \quad \text{and} \quad x \underset{E}{\leq} B \quad .$$

Thus by the induction hypothesis

$$x \underset{E}{\leq} \text{SUPP}(x,A) \quad \text{and} \quad x \underset{E}{\leq} \text{SUPP}(x,B)$$

and therefore,

$$x \underset{E}{\leq} \text{MIN}(\text{SUPP}(x,A),\ \text{SUPP}(x,B)) \ ,$$

as required.

<u>Step 7.3</u>. y has the form $b\,x + c$, where b is a number, $b < 1$, c does not contain x, and c is quasi-linear in V.

Since by hypothesis we have

$$x \underset{E}{\leq} b\ x + c$$

it follows from Lemma 1.4, 1.6,

$$x(1-b) \underset{E}{\leq} c \ ,$$

$$x \underset{E}{\leq} \frac{c}{1-b} = \text{SUPP}(x,y)$$

as required.

<u>Case 7.5.2</u>. y has the form $x + c$, where $c < 0$.

Since by hypothesis we have

$$x \underset{E}{\leq} x + c \quad ,$$

it follows from Lemma 1.4 that

$$0 \underset{E}{\leq} c < 0$$

in contradiction to Lemma 1.3. This means that the hypothesis of Lemma 1.5,

$$E \neq \emptyset$$

is false. That is $E = \emptyset$ (E is empty) and hence the desired conclusion

$$x \underset{E}{\leq} \text{SUPP}(x,y) = -\infty$$

is (vacuously) true.

<u>Theorem 9</u>. If J is quasi-linear in V, and $L \subseteq V$, then a call SUP(J,L) to the recursive algorithm SUP halts after a finite number of steps.

<u>Proof</u>. The following informal proof can be made more rigorous by multiple induction (not recursive induction, which itself would require termination).

A call, SUP(J,L), results in a computation tree, which depends on the nature of J, L, and the contents of S. The nodes of the tree are calls to the functions SUP, INF, SUPP, INFF, SIMP, MAX, MIN, and other non-recursive functions.

When $J \in V$ and $J \notin L$, the call SUP(J,L) exits through Step 2.2 by recalling SUP(b, $L \cup \{J\}$) where b is an entry from S, namely UPPER(S,J). There can be at most n such calls from Step 2.2 on any branch of the computation tree, where n is the number of members of V, because such a call is made only if $J \notin L$ and since J is then added to L. Similarly there can be at most n calls to INF from Step 2.2.

Between any two such "Step 2.2 calls", SUP(A,L') and SUP(B,L"), (of step 2.2 calls in the algorithm INF) there can be at most m calls to SUP or INF where m is the number of symbols in A, because all other exits from A, SUP(A',LL) or INF(A',LL) are on formulas A' which have fewer symbols than does A. (Lemma 11 below) Also the algorithms SIMP, MAX, and MIN halt after a finite number of steps since they are not recursive, and so also do SUPP and INFF by Theorem 10. Thus any branch of the tree is finite and hence the computation for SUP(J,NIL) terminates in a finite number of steps.

<u>Theorem 10</u>. The algorithms SUPP and INFF each terminate after a finite number of steps.

<u>Proof</u>. We give a proof for SUPP only, the one for INFF is similar. Note that only Steps 4,5, and 6 have recursive calls and they only to SUPP itself. Steps 4 and 5 eliminate a symbol (either "max" or "min") from the first argument, and Step 6 pulls only existing "max"'s and "min"'s to the

front to be eliminated by Steps 4 and 5. Thus if there are k "max"'s and "min"'s in y, then there can be at most $2k+1$ nested calls to SUPP before termination.

Lemma 11. If J is quasi-linear in V, and $L \subseteq V$, and the computation of SUP(J,L) does not involve a call to Step 2.2 in the algorithm SUP or Step 2.2 in the algorithm INF, then

.1 SUP(J,L) = J or $+\infty$, and

.2 each subcall SUP(D,LL) or INF(D,LL) is on a formula D having fewer symbols than does J.

Proof. Part .1 follows from the fact that only Steps 2.2 and 10 changes the formula J. The other steps either return J itself or break it down into its component parts. A formal proof of this can be made using induction on the number of symbols in J.

For the proof of .2, we note that in all but Steps 2.2, 6.2.2, and 7.2 the subcalls to SUP and INF are on formulas D with fewer symbols than J. For example, in Step 3 the subcall is to INF(A,L) and A has one fewer symbol than does ("-"A). Similarly, in Step 6.2.1 the two subcalls are to SUP(B, $L \cup \{A'\}$), and SUP(A,L); and each of B and A have fewer symbols than does ("+" A B).

Since by hypothesis the computation of SUP(J,L) does not involve a call to Step 2.2, it follows from .1 that $B' = B$ in Step 6 and $J' = J$ in Step 7, and hence Steps 6.2.2 and 7.2 cannot arise.

Remarks.

Our procedure for proving Presburger formulas with only universal quantification, which is described in Section 2, utilizes the algorithms SUP and INF to compute $\sup_S$ and $\inf_S$. Since by Theorem 5,

$$(8) \qquad \mathrm{INF}(J,\mathrm{NIL}) \leq \inf_S J \ , \qquad \sup_S J \leq \mathrm{SUP}(J,\mathrm{NIL})$$

it follows that this procedure is sound.

It can be shown by example that we may not have equality in (8). For example, if

$$S = \{(0 \leq k \leq J)(0 \leq J \leq L)(0 \leq 1 \leq (J + (4-k))\} \quad ,$$

the algorithm SUP gives

$$\mathrm{SUP}(k,\mathrm{NIL}) = +\infty \quad ,$$

whereas, $J \leq L \leq J + 4-k$, $0 \leq 4-k$, $k \leq 4$,

$$\sup_S k = 4 \quad .$$

However, if S is in "natural form" (see definition below) we conjecture that

$$(9) \qquad \mathrm{INF}(J,\mathrm{NIL}) = \inf_S J \ , \qquad \mathrm{SUP}(J,\mathrm{NIL}) = \sup_S J \quad .$$

If this conjecture is true then our procedure is a <u>decision</u> procedure for Presburger formulas with only universal quantification, because our procedure automatically constructs S in a natural form.*

<u>Definition</u>. A formula S is said to be in "natural form" if it is a disjunction

$$S \equiv S_1 \vee S_2 \vee \ldots \vee S_k \quad ,$$

where each of the S_i is a conjunction

$$S_i = \bigwedge_{j=1}^{n} (a_{ij} \leq x_j \leq b_{ij}) \quad ,$$

and where, for each j, a_{ij} and b_{ij} are quasi-linear in $x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_n$, and each S_i has the redundant property defined as follows.

Each variable in S_i is represented by a conjunct $(a \leq x \leq b)$ in S_i and this contains all the information about x that S_i has. For example, if

$$a \leq x \leq b \quad \text{and} \quad f(x) \leq y \leq d$$

are conjuncts of S_i and "max" and "min" do not appear in f(x), then the information in $(f(x) \leq y)$ is already contained in $(a \leq x \leq b)$. It might be in the form

*Robert Shostak (Private Communication) has recently shown that if S is in natural form and J is a variable, then the equalities hold in formulas (8), and hence that this is indeed a decision procedure.

$$a \leq x \leq f^{-1}(y) \quad ,$$

or

$$a \leq x \leq (\min f^{-1}(y)\ c)$$

etc. Similarly if $f(x)$ has the form $(\max g(x)\ c)$ then $f(x) \leq y$ is equivalent to $(g(x) \leq y \wedge c \leq y)$, and again in this case we require that the information $x \leq g^{-1}(y)$ to already be contained in $a \leq x \leq b$.

General Presburger formulas.

As mentioned earlier, it is not at all clear whether this procedure can be extended to handle general Presburger formulas with both existential and universal quantification. Luckily, the examples we have so far encountered in program validation [10] have all had only universal quantification.

A Presburger formula with only existential quantification can be equally well handled (assuming that our conjecture (9) is true) but there has been as yet not use for that in our applications.

5. Examples

We will show some examples where the sup-inf procedure is used to prove some theorems and non-theorems in Presburger arithmetic. We also show some examples where only the SUP and INF algorithms are exercised.

The first few examples are given in great detail but later ones omit some or all of the intermediate steps.

Here we use the notation $\{x: a\ b\}$ to denote $a \le x \le b$. In the examples we sometimes indicate in the right hand margin the rule number (Step No.) of SUP, INF, SUPP, or INFF being employed.

1. Example (from page 1). Th. $(x < 1 \rightarrow x < 2)$.

$$\sim \text{th. } (x < 1 \wedge 2 \le x)$$

$$(x \le 0 \wedge 2 \le x)$$

$$(2 \le x \le 0)$$

$$S = (\{x: 2\ \ 0\})$$

		Rule no.
a)	$\text{SUP}(x,\text{NIL}) = 0$	2.2
	$\text{SUP}(0,\{x\}) = 0$	1
	$\text{SUPP}(x,0) = 0$	1
b)	$\text{INF}(x,\text{NIL}) = 2$	2.2
	$\text{INF}(2,\{x\}) = 2$	1
	$\text{INFF}(x,2) = 2$	1

Since $0 < 2$, the theorem is true.

2. Example (from Page 2) Th. $(5x < 11 \rightarrow 7x < 16)$

$$\sim \text{th.}\ (5x < 11 \wedge 16 \leq 7x)$$
$$(5x \leq 10 \wedge 16 \leq 7x)$$
$$(x \leq 2 \wedge \frac{16}{7} \leq x)$$

$$S = (\{x: \frac{16}{7} \quad 2\})$$

$$SUP(x,NIL) = 2 < \frac{16}{7} = INF(x,NIL).$$

So the Theorem is true.

3. Example. th(?). $(5x < 16 \rightarrow 7x < 16)$

$$\sim \text{th.}\ (5x < 16 \wedge 16 \leq 7x)$$
$$(5x \leq 15 \wedge 16 \leq 7x)$$

$$S = (\{x: \frac{16}{7} \quad 3\})$$

Thus $SUP(x,NIL) = 3$, $INF(x,NIL) = \frac{16}{7}$.

Since $[INF(x,NIL), SUP(x,NIL)] = [\frac{16}{7}, 3]$ contains an integer, the supposed theorem is false.

4. Example (Formula (3), Page 4, with x,y,z for x_1, x_2, x_3).

$$\text{Th.}\ (2y + 3 \leq 5z \wedge z \leq x - y \wedge 3x \leq 5 \rightarrow 2y \leq 3)$$

From formula (8'), Page 4, we get

$$S = (\{x: z+y\ \tfrac{5}{3}\}$$
$$\{y: 2\ \ \min(\tfrac{5}{2}z - \tfrac{3}{2},\ \ x-z)\}$$
$$\{z: \tfrac{2}{5}y + \tfrac{3}{5}\qquad x-y\})$$

	Rule no.
a) $SUP(x,NIL) = \frac{5}{3}$	2.2
$SUP(\frac{5}{3},\{x\}) = \frac{5}{3}$	1
$SUPP(x,\frac{5}{3}) = \frac{5}{3}$	1
b) $INF(x,NIL) = \frac{17}{5}$	2.2
$INF(z+y,\{x\}) = \frac{17}{5}$	6
$B' = INF(y,\{x\ z\}) = 2$	2.2
$INF(2,\{x\ y\ z\}) = 2$	1
$INFF(y,2) = 2$	1
$J' = (z+2)$	6.2.2
$INF(z+2,\{x\}) = \frac{7}{5} + 2 = \frac{17}{5}$	6
$B' = INF(2,\{x\ z\}) = 2$	1
$INF(z,\{x\}) = \frac{7}{5}$	2.2
$INF(\frac{2}{5}y + \frac{3}{5}, \{x\ z\}) = \frac{4}{5} + \frac{3}{5} = \frac{7}{5}$	6
$B' = INF(\frac{3}{5}, \{x\ y\ z\}) = \frac{3}{5}$	1
$INF(\frac{2}{5}y, \{x\ z\}) = \frac{4}{5}$	5.1
$INF(y,\{x\ z\}) = 2$	2.2
$INF(2,\{x\ y\ z\}) = 2$	1
$INFF(y,2) = 2$	1
$INFF(z,\frac{7}{5}) = \frac{7}{5}$	1
$INFF(x,\frac{17}{5}) = \frac{17}{5}$	1

Thus as stated on Page 5

$$\mathrm{SUP}(x,\mathrm{NIL}) = \frac{5}{3} < \frac{17}{5} = \mathrm{INF}(x,\mathrm{NIL}) ,$$

and the theorem is true.

5. Example. $S = (\{j\colon 0 \quad k\}\{k\colon 0 \ j+L\}\{L\colon 0 \quad 2-j\})$

Clearly $\sup_S k = 2$, and also $\mathrm{SUP}(k,\mathrm{NIL}) = 2$. But if we ignore Rule 6 of the algorithm SUP, we get $+\infty$ as follows, which is incorrect.

	Rule no.
$\mathrm{SUP}(k,\mathrm{NIL}) = \infty$	2.2
$\mathrm{SUP}(j+L,\{k\}) = k+2$	7
$\mathrm{SUP}(j,\{k\}) = k$	2.2
$\mathrm{SUP}(k,\{j\,k\}) = k$	2.1
$\mathrm{SUPP}(j,k) = k$	7.1
$\mathrm{SUP}(L,\{k\}) = 2$	2.2
$\mathrm{SUP}(2-j,\{k\,L\}) = 2+0 = 2$	7
$\mathrm{SUP}(2,\{k\,L\}) = 2$	1
$\mathrm{SUP}(-j,\{k\,L\}) = 0$	3
$\mathrm{INF}(j,\{k\,L\}) = 0$	2.2
$\mathrm{INF}(0,\{j\,k\,L\}) = 0$	1
$\mathrm{INFF}(j,0) = 0$	1
$\mathrm{SUPP}(L,2) = 2$	1
$\mathrm{SUPP}(k,k+2) = \infty$	7.5.3

6. Example. $S = (\{j\colon 0\ L\}\ \{k\colon j\ j\}\ \{L\colon 0(2j-3k)\})$

	Rule no.
a) SUP(j,NIL) = 0	
b) INF(j,NIL) = 0	
c) SUP(k,NIL) = $+\infty$	2.2
SUP(j,{k}) = $+\infty$	2.2
SUP(L,{j k}) = $2j-3k$	2.2
SUP($2j-3k$,{j k L}) = $2j-3k$	6,3,5.1,2.1,6.1,5.1,2.1
SUPP(L,$2j-3k$) = $2j-3k$	7.1
SUPP(j,$2j-3k$) = $+\infty$	7.4
SUPP(k,$+\infty$) = $+\infty$	1
d) INF(k,NIL) = 0	
e) SUP(L,NIL) = 0	
f) INF(L,NIL) = 0	

Actually, for this example,

$$\sup_S j = \sup_S k = \sup_S L = 0 \quad ,$$
$$\inf_S j = \inf_S k = \inf_S L = 0 \quad ,$$

whereas the SUP algorithm gave

$$\text{SUP}(k,\text{NIL}) = +\infty \quad .$$

As was stated in Section 4 we only expect equality between sup and SUP when S is in natural form.

We now naturalize S, and again compute these quantities (in Example 7 below)

7. Example.

$$\begin{array}{lll} S = (\{j: & \max(\max(0,k), \frac{L}{2} + \frac{3}{2}k) & \min(k,L)\} \\ \{k: & j & \min(j, \frac{2}{3}j - \frac{L}{3}\} \\ \{L: & j & 2j - 3k\}) \end{array}$$

<u>Rule no.</u>

a) SUP(j,NIL) = 0

b) INF(j,NIL) = 0

	Rule no.
c) $\mathrm{SUP}(k,\mathrm{NIL}) = 0$	2.2
$\mathrm{SUP}(\min(j,\frac{2}{3}j - \frac{L}{3}),\{k\}) = \min(k,\frac{1}{3}k)$	9
$\mathrm{SUP}(j,\{k\} = k$	2.2
$\mathrm{SUP}(\min(k,L),\{j\ k\}) = \min(k,2j-3k)$	9
$\mathrm{SUP}(k,\{j\ k\}) = k$	2.1
$\mathrm{SUP}(L,\{j\ k\}) = 2j-3k$	2.2
$\mathrm{SUP}(2j-3k,\{j\ k\ L\}) = 2j-3k$	6,3,5,2.1
$\mathrm{SUPP}(L,2j-3k) = 2j-3k$	7.1
$\mathrm{SUPP}(j,\min(k,2j-3k)) = k$	5
$\mathrm{SUPP}(j,k) = k$	7.1
$\mathrm{SUPP}(j,2j-3k) = \infty$	7.5.3
$\mathrm{SUP}(\frac{2}{3}j - \frac{L}{3}, \{k\}) = \frac{1}{3}k$	6
$B' = \mathrm{SUP}(-\frac{L}{3},\{j\ k\}) = -\frac{j}{3}$	3
$\mathrm{INF}(\frac{L}{3},\{j\ k\}) = \frac{j}{3}$	5.1
$\mathrm{INF}(L,\{j\ k\}) = j$	2.2
$\mathrm{INF}(j,\{j\ k\ L\}) = j$	2.2
$\mathrm{INFF}(L,j) = j$	1
$J' = \mathrm{SIMP}(\frac{2}{3}j - \frac{j}{3}) = \frac{1}{3}j$	6.2.2
$\mathrm{SUP}(\frac{1}{3}j,\{k\}) = \frac{1}{3}k$ (as before)	
$\mathrm{SUPP}(k,\min(k,\frac{1}{3}k)) = 0$	5
$\mathrm{SUPP}(k,k) = +\infty$	7.5.3
$\mathrm{SUPP}(k,\frac{1}{3}k) = 0$	7.3

d) INF(k,NIL) = 0

e) SUP(L,NIL) = 0

f) INF(L,NIL) = 0 .

Thus they are all now 0 as desired.

8. Example. S = ({k: 0 j} {j: 0 L} {L: 0 j-k+4}).

S is not in natural form.

$$\mathrm{Sup}_S\ k = 4, \quad \text{but} \quad \mathrm{SUP}(k,\mathrm{NIL}) = +\infty .$$

If we naturalize S, getting

$$\begin{aligned} S = (\{k:&\ 0 \quad \min(j, j+4)\} \\ \{j:&\ \max(k, L+k-4) \quad L\} \\ \{L:&\ j \qquad\qquad j-k+4\}) \end{aligned} ,$$

then

$$\mathrm{SUP}(k,\mathrm{NIL}) = 4$$

as desired.

9. Example. S = ({j: 0 k+L} {k: 0 5} {L: 0 k}) .

	Rule no.
a) SUP(j,NIL) = 10	2.2
SUP(k+L,{j}) = 10	6
B' = SUP(L,{j k}) = k	2.2
SUP(k,{j k L}) = k	2.1
SUPP(L,k) = k	7.1
J' = k+k = 2k	6.2.2
SUP(2k,{j}) = 10	5.1
SUP(k,{j}) = 5	2.2
SUP(5,(j k)) = 5	1
SUPP(k,5) = 5	1
SUPP(j,10) = 10	1

b) INF(j,NIL) = 0

c) SUP(k,NIL) = 5

d) INF(k,NIL) = 0

e) SUP(L,NIL) = 5

f) INF(L,NIL) = 0

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

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's algorithm. Summer Inst. for Symbolic Logic, Cornell U., 1957. 215-233.

3. Presburger, M. Uber die Vollstandigkeit eines genissen Systems der Arithmetik ganzer Zahlen, in Welchem die Addition als einzige Operation hervortritt, Sprawozdanie z I Kongresu Matematykow Krajow Slowcanskich Warszawa, 1929, pp. 92-101.

4. Wang, Hao. Toward Mechanical Mathematics, IBM J. Res. Dev. 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. King, J. A program verifier. Ph.D. thesis, Carnegie-Mellon Univ., Pittsburgh, 1969.

7. Deutch, L.P. An Interactive Program Verifier, Ph.D. Thesis, University of California, Berkeley, 1973.

8. Waldinger, R.J. and Levitt, K.N. Reasoning about Programs, Artif. Jour. 5(1974), 235-316.

9. Igarashi, S., London, R.L., and Luckham, D.C. Automatic program verification 1: a logical basis and its implementation. Stanford AI Memo 200, May 1973 and USC Information Sciences Institute Report ISI/RR-73-11, May 1973.

10. 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.

11. Suzuki, N. Automatic Program Verification II: Verifying programs by algebraic and logical reduction. Stanford AI Memo AIM-255, Dec. 1974.

12. Bledsoe, W.W. Program Correctness, The University of Texas at Austin Mathematics Department Memo ATP 14 Jan. 1974.

13. Bledsoe, W.W. and Tyson, M. Typing and proofs by cases in program verification (working title), The University of Texas at Austin Mathematics Department Memo ATP 15 (forthcoming).

14. Bledsoe, W.W. and Bruell, P. A Man-Machine theorem-proving System. *Artif. Intell.* 5(1974), 51-72.

15. Hearn, A.C. Reduce 2: A system and language for algebraic manipulation, *Proceedings of the Second Symposium on Symbolic and Algebraic Manipulation,* ACM, 1971, 128-133.

16. Burstall, R.M. Proving properties of programs by structural induction. *Computer J.* 12, 1 (Feb. 1969), 41-48.

17. McCarthy, John. A basis for mathematical theory of computation. In *Computer programming and formal systems,* P. Brafford and D. Hirshberg (Eds.) North-Holland Publ. Co., Amsterdam, 1963, 33-70.