AN ALGORITHM FOR CHECKING PL/CV ARITHMETIC INFERENCES

77-326

Tat-Hung Chan

Department of Computer Science
Cornell University
Ithaca, N.Y. 14853

AN ALGORITHM FOR CHECKING PL/CV ARITHMETIC INFERENCES

77-326

Tat-Hung Chan

Department of Computer Science
Cornell University
Ithaca, N.Y. 14853

Abstract: This paper describes the operation and implementation of the arithmetic proof rule for the quantifier free integer arithmetic used in the PL/CV 2 program verification system. The general arithmetic satisfiability problem underlying the rule is shown to be NP complete.

This work was supported in part by NSF grants MCS 76-14293 and MCS 75-09433.

§1 Introduction

Arithmetic operations constitute an indispensible component of algebraic computing languages. Consequently, arithmetic reasoning plays a vital role in verification schemes for programs written in these languages. The most indispensible type of arithmetic reasoning involves only the integers.

The PL/CV 2 verification system, Constable and O'Donnell [78], and Constable and Johnson [78], provides an integer arithmetic which is a fragment of a constructive quantifier free theory of discrete ordered integral domains.+ A verification rule, called the arithmetic rule, allows the conclusion of a disjunction of arithmetic relations from a conjunction of arithmetic relations provided the disjunction can be deduced from the conjunction in the underlying quantifier free theory of arithmetic (in a restricted manner to be described later). This paper describes the operation and implementation of the arithmetic rule.

The paper is organized as follows. Section 2 summarizes the underlying arithmetic axioms and formulates the arithmetic rule in terms of quantifier-free proofs using these axioms. Section 3 consi the implementation of the rule, i.e. the question of how to verify its applicability. This is essentially a proof existence problem, which will be shown to be equivalent to the satisfiability problem for a set of arithmetic relations over constants and monic linear

+ Although the PL/CV 2 arithmetic theory is constructive (even Intuitionistic), since we are dealing only with quantifier free proofs involving computable relations, all the relevant results from classical logic remain applicable (see Kleene [52]).

univariate polynomials. In proving this equivalence, we shall introduce the closely related problem of the satisfiability of directed graphs with edges weighted by the integers, and make use of the lemma that such a graph is satisfiable if and only if it contains no cycle[+] of positive weight. Besides being a useful tool in the theoretical discussion, this graph-theoretic result will play an important role in the actual implementation. In Section 4, we establish the NP-completeness of the arithmetic satisfiability problem of Section 3. The problem is then represented as a tree of directed, weighted graphs, and an algorithm for its solution by searching for a satisfiable leaf is developed. Section 5 contains an account of the actual implementation of the arithmetic rule and some examples of its use. We conclude with some remarks on possible extensions in Section 6.

[+] In this paper, cycles and paths in directed graphs are understood to be directed.

§2 The Arithmetic Rule

As mentioned above, PLCV 2 admits four groups of arithmetic axioms. These are

(I) the ring axioms,

(II) the discrete linear order axioms,

(III) the relation definition axioms, and

(IV) the monotonicity axioms.

Groups I axiomatizes the properties of the integers as a commutative ring $(\mathbb{Z},+,\cdot,0,1)$. Group II axiomatizes the properties of the discrete linear order "less than" ($<$), while Group III defines $>$, $\le$ and $\ge$ in terms of $<$ and the logic connectives. Finally, Group IV axioms allow the combination of two arithmetic relations by the binary arithmetic operations; they are all variants of the four basic forms:

(addition)	$x\ge y$ & $z\ge w \Rightarrow x+z\ge y+w$
(subtraction)	$x\ge y$ & $z\le w \Rightarrow x-z\ge y-w$
(multiplication)	$x\ge 0$ & $y\ge z \Rightarrow xy\ge xz$
(factoring)	$x>0$ & $xy\ge xz \Rightarrow y\ge z$

A complete list of all the arithmetic axioms can be found in the append

The arithmetic rule will be concerned only with arithmetic propositions, which are quantifier-free propositions whose atoms are arithmetic relations. Arithmetic relations in PLCV 2 are of the form $\tau_1 \rho \tau_2$, where τ_1,τ_2 are polynomial expressions in program variables (possibly subscripted), logic variables, and function designators+, and ρ is one of the six relational operators $<,\le,=,\ne$, $\ge$ and $>$. By repeated applications of the ring axioms, any polynomial expression τ can be proved equal to some polynomial of the form

+ For simplicity, this paper considers only expressions involving simple variables. A discussion of subscripted variables and function designators can be found in Constable and Johnson [78].

$c+\pi$ ($c\neq 0$), π, or c, where c is an integer constant and π is a constant-free polynomial in some chosen canonical form. For appropriately chosen canonical forms, the expression $c+\pi$, π or c will also be the canonical form of τ. In the remainder of this section, we shall consider two polynomial expressions to be the same if they have the same canonical form.

Obviously, applications of the axioms of Groups I, II and III can only produce conclusions involving the same constant-free polynomials as occur in the antecedents. Applications of the monotonicity axioms introduce new constant-free polynomials in some, but not all, cases. Prominent among those which do not are the additions in which one relation has constant comparands, e.g.

$$x^2 \geq y \ \& \ \ 2>3 \ \ \Rightarrow x^2-2>y-3$$

In the remainder of this paper, we refer to such additions as <u>trivial monotonicity</u>, and all other applications of the monotonicity axioms as <u>nontrivial monotonicity</u>.

Thus, a quantifier-free proof using only propositional calculus, the PL/CV2 equality axioms (reflexivity, symmetry, transitivity and substitution of comparands+), the arithmetic axioms of the first three groups and trivial monotonicity essentially involves only those constant-free polynomials that occur in the antecedents, and is therefore of limited complexity. We call such a proof a <u>restricted arithmetic proof</u>. If we further restrict the hypothesis and the

+ This is a restriction of the substitution axiom which disallows the substitution of only part of a comparand. For example, from $x\neq x*y$ & $x=z$ it can justify $z\neq x*y$ but not $z\neq z*y$.

conclusion to be, respectively, a conjunction and a disjunction of arithmetic relations, then the decision problem for the existence of a restricted arithmetic proof is reducible to an integer satisfiability problem, solvable by graph algorithms. This reduction, which will be formulated and proved rigorously in the next section, is exploited in the design of the arithmetic verification rule, which is intended to embody as many applications of the arithmetic axioms as possible without rendering the validity checking (i.e. proof existence problem impossible. Specifically, the arithmetic rule can be used to justify the deduction of a conclusion C from a hypothesis H if and only if

(1) H is a conjunction of arithmetic relations,

(2) C is a disjunction of arithmetic relations, and

(3) there exists a restricted arithmetic proof of C from M, where M is either H, or H enhanced by one application of nontrivial monotonicity to be specified by the user.

In connection with nontrivial monotonicity, it should be noted that factoring is achieved by "syntactic division", i.e. in order to divide out a factor x from a relation, the user must exhibit each comparand explicitly as a product of two factors, one of which must be x (unless the comparand is either 0, or is x itself). The effect of this restriction is to shift the burden of polynomial division from the verifier to the user. Also note that the set of constant-free polynomials involved becomes fixed after the single application of nontrivial monotonicity (if specified).

Examples. The argument

$$x+y>z,\ 2x\geq z \Rightarrow 3x+y\geq 2z-1$$

can be justified by one invocation of the arithmetic rule, whereas the argument

$$x>y,\ y>0,\ z>w,\ w>0 \Rightarrow xz>yw$$

requires three invocations:

$z>w,\ w>0 \Rightarrow z>0$ (transitivity)

$x>y,\ z>0 \Rightarrow xz>yz$ (multiplication by z)

$z>w,\ y>0,\ xz>yz \Rightarrow xz>wy$ (multiplication by y followed by transitivity).

§3 Reduction of the Validity Checking Problem.

The module of PLCV 2 which checks the applicability of the arithmetic rule to a verification step is called the arithmetic checker. Nontrivial monotonicity aside, its function is to decide whether there exists a restricted arithmetic proof for a disjunction of arithmetic relations C from a conjunction of arithmetic relations H. For convenience, we shall abbreviate the term restricted arithmetic proof to A-proof. Extending this terminology, we say that a proposition Q is A-provable from a proposition P, denoted $P \vdash_A Q$, if there exists an A-proof of Q from P, and that P is A-contradictory if a contradiction (i.e. a proposition of the form $Q \wedge \neg Q$) is A-provable from P. We now reduce the A-proof existence problem to an integer satisfiability problem.

Th^{m}1. Let P, Q be arithmetic propositions. Then $P \vdash_A Q \Leftrightarrow P \wedge \neg Q$ is A-contradictory.

Proof: Standard result in logic.

Thus it suffices for the arithmetic checker to decide whether the conjunction of relations $H \wedge \neg C$ is A-contradictory.

Th^{m}2. Let P be a conjunction of arithmetic relations, and let P_C be the conjunction of the same relations with all comparands converted to canonical form. Then P is A-contradictory $\Leftrightarrow$ P_C is A-contradictory.

Proof: Follows from the fact that $P \vdash_A P_C$ and $P_C \vdash_A P$ by propositional calculus, the equality axioms and the ring axioms.

The next theorem allows us to simplify the problem by consider the distinct constant-free polynomials in P_C as "atomic". Intuitive this abstraction, whereby we ignore relations that may exist among the constant-free polynomials by virtue of their internal structures is allowable because A-proofs, being deprived of the use of nontrivi monotonicity, are not powerful enough to exploit these structures anyway. Note that, as mentioned in the previous section, it is precisely this limitation of the power of A-proofs which makes their existence problem easier (yet still NP-complete, as we shall see lat

Thus let P_C be a conjunction of arithmetic relations with canonical form comparands, and let P_A be P_C with all distinct non-zero constant-free parts of the comparands replaced by new, distinct variables. For example, if P_C is

$$1+2x = z+xy \ \& \ -3-2x > -4+z+xy$$

then P_A is

$$1+u = v \ \& \ -3+w > -4+v.$$

Then we have

Th^{m}3. P_C is A-contradictory $\Leftrightarrow$ P_A is A-contradictory.

<u>Proof</u>: "$\Leftarrow$" Take an A-proof of contradiction from P_A. Replace all occurrences in the proof of those variables that are in P_A by their corresponding constant-free polynomials, and leave occurrences of variables that are not in P_A unchanged. The result is a proof in which justification by the arithmetic and equality axioms and by the various rules of inference remain valid, since the relevant structures of the comparands and the propositional structures are left intact by the replacements. The only change is that an appeal to P_A becomes an appeal to P_C. Since a contradiction in the old proof is transformed to a contradiction in the new proof, we have shown that P_C is A-contradictory.

"$\Rightarrow$" Again we proceed by "proof-massaging".
Take an A-proof of contradiction from P_C, and compute the canonical form of each comparand. There are several possibilities for the canonical form, and we transform the comparands accordingly.

(T1) The canonical form is c for some integer c:
the original comparand is left unchanged.

(T2) The canonical form is π or $c+\pi$ where c is an integer $\neq 0$, and π is the non-zero constant-free part of some comparand in P_C: replace the comparand by x_π or $c+x_\pi$ where x_π is the variable corresponding to π.

(T3) The canonical form is π or $c+\pi$ as in (T2) except that π is not the constant-free part of any comparand in P_C: replace each occurrence of any variable in the comparand by some fixed variable y not in P_A.

Note that both (T1) and (T3) leave the tree structure of the comparand intact.

We next manipulate the intermediate text resulting from comparand transformation into a valid A-proof from P_A. This is achieved by a tedious case analysis of the justification of each st in the old proof. In the following analysis, x' denotes the transformation of comparand x.

(1) Propositional calculus and equality axioms. The justificatio remains valid because if comparands x,y are identical, then s are x', y', so that the structures of the arithmetic proposit are preserved.

(2) Appeal to P_C. Transformed to an appeal to P_A.

(3) Arithmetic axioms.

(i) Ring axioms. The proposition is an equality, the two sides of which necessarily have the same canonical form. Hence either they both transform under (T2) to the same new comparand, in which case the equality is an identity justified by the reflexivity of equality, or they are le unchanged except for possible replacement of variables u (T1) or (T3), so that the original ring axiom still hold

(ii) Irreflexity, trichotomy, transitivity, and relation definitions. The original justification remains valid for the same reason as in (1).

(iii) Discreteness ($\neg x<y<x+1$). Clearly the same transformatio applies to both comparands x and x+1. If this transformation is (T1) or (T3), then discreteness remains a vali justification. It remains to consider transformation by (T2). If x and x+1 transform to x_π and $1+x_\pi$ for some constant-free polynomial π in P_C, insert in front of the

transformed proposition the steps

$\neg(x_\pi < y' < x_\pi+1)$	discreteness
$x_\pi+1 = 1+x_\pi$	commutativity

If x and x+1 transform to $-1+x_\pi$ and x_π, insert steps

$\neg(-1+x_\pi<y' < (-1+x_\pi)+1)$ discreteness

$\vdots$

proof that $(-1+x_\pi)+1=x_\pi$ (by ring axioms & equality axioms)

$\vdots$

Finally let x and x+1 transform to $c+x_\pi$ and $\overline{c+1}+x_\pi$, $c\neq 0$ or -1. Insert steps

$\neg(c+x_\pi<y'<(c+x_\pi)+1)$ discreteness

$\vdots$

proof that $(c+x_\pi)+1=\overline{c+1}+x_\pi$ (by ring axioms & equality axioms)

In all three cases, use "comparand substitution" as the justification of the transformed step.

(iv) Trivial monotonicity. The proposition is in one of two formats:

$x\rho y \wedge z\sigma w \rightarrow x+z\ \tau\ y+w$ or

$x\rho y \wedge z\sigma w \rightarrow x+z\ \tau\ y+w \wedge x+w\ \nu\ y+z$

where one of the antecedents, say $z\sigma w$, has constant comparands. Thus we know that z and w are left unchanged (T1), while x, x+z and x+w have the same constant-free part and so undergo similar transformation, and similarly for y, y+w and y+z. If neither x nor y is transformed by (T2), then we know that the original justification remains valid. Now suppose at least one of x,y is transformed by (T2). Then the original proposition is replaced by the sequence:

$$\begin{array}{l}
x'\rho y' \wedge z\sigma w \to x'+z \; \tau \; y'+w \text{ (trivial monotonicity)} \\
\vdots \\
\text{proof that } x'+z=(x+z)' \text{ (if } x \text{ transformed by (T2))} \\
\vdots \\
\text{proof that } y'+w=(y+w)' \text{ (if } y \text{ transformed by (T2))} \\
\vdots \\
x'\rho y' \wedge z\sigma w \to (x+z)' \; \tau (y+w)' \text{ by comparand substitution.}
\end{array}$$

Note that $z'=z$ and $w'=w$ since they are constants, and that if x (respectively y) is not transformed by (T2), then $x'+z=(x+z)'$ (respectively, $y'+w=(y+w)'$) already. Similarly for the second format.

Since contradictory propositions are transformed to contradictory propositions, P_A is A-contradictory.

So far, we have simplified the problem to deciding whether a conjunction of arithmetic relations $R_1 \wedge \cdots \wedge R_m$ is A-contradictory, where each R_i has comparands of the form 0, c, x or c+x for some nonzero constant c and some variable x. From the distributive laws for $\wedge$ and $\vee$, trivial monotonicity, and the A-provable equivalences:

$$\begin{array}{lll}
x>y & \vdash\dashv_A & x\geq y+1, \\
x<y & \vdash\dashv_A & y\geq x+1, \\
x=y & \vdash\dashv_A & x\geq y \wedge y\geq x, \\
x\neq y & \vdash\dashv_A & x\geq y+1 \vee y\geq x+1, \text{ and} \\
x\leq y & \vdash\dashv_A & y\geq x,
\end{array}$$

it follows that $R_1 \wedge \ldots \wedge R_m \vdash\dashv_A S_1 \vee S_2 \vee \cdots \vee S_n$ where each S_i is a conjunction of m' ($m'\geq m$) relations of the form

$$\begin{Bmatrix} 0 \\ x \end{Bmatrix} \geq \begin{Bmatrix} c \\ y\,[+d] \end{Bmatrix}$$

for variables x,y and constants c,d, $d\neq 0$. Here, $n=2^k$ where k= numbe of $\neq$-relations among the R_i's. From this equivalence we immediately obtain, by classical logic, the following lemmas:

Lemma 1: $R_1 \wedge \cdots \wedge R_m$ is A-contradictory $\Leftrightarrow$ $S_1 \vee \cdots \vee S_n$ is A-contradictory.

Lemma 2: $R_1 \wedge \cdots \wedge R_n$ is unsatisfiable $\Leftrightarrow$ $S_1 \vee \cdots \vee S_n$ is unsatisfiable.

The following are equally obvious:

Lemma 3: $S_1 \vee \cdots \vee S_n$ is A-contradictory $\Leftrightarrow$ each S_i is A-contradictory.

Lemma 4: $S_1 \vee \cdots \vee S_n$ is unsatisfiable $\Leftrightarrow$ each S_i is unsatisfiable.

Thus, provided we can establish

Lemma 5: S_i is A-contradictory $\Leftrightarrow$ S_i is unsatisfiable,

we have achieved the last step in our reduction:

Th^{m}4: $R_1 \wedge \cdots \wedge R_n$ is A-contradictory $\Leftrightarrow$ $R_1 \wedge \cdots \wedge R_n$ is unsatisfiable.

Proof of Lemma 5: We prove this lemma by introducing a directed, weighted graph associated with S_i. This graph will play an important role in the satisfiability algorithm actually implemented. Thus let G be a graph whose vertices are the variables in S_i plus an extra vertex for 0 if there is any constant comparand. For each relation

$$\begin{Bmatrix} x \\ 0 \end{Bmatrix} \geq \begin{Bmatrix} c \\ y[+d] \end{Bmatrix}$$

in S_i, there is an edge from x or 0 to 0 or y with weight c or 0 or d as the case may be. The weight of a path in this graph is the algebraic sum of the weights of the edges in the path. A positive cycle is a cycle with positive weight. A graph G is satisfiable if there exists an assignment of integers $\bar{u}$ to the vertices u of G, such that for each edge $u \to v$ of weight m in G, we have $\bar{u} \geq \bar{v}+m$.

We now show that the following statements are equivalent:

(1) S_i is A-contradictory,

(2) S_i is unsatisfiable,

(3) The graph associated with S_i is unsatisfiable,

(4) The graph associated with S_i has a positive cycle.

(1) $\Rightarrow$ (2). Obvious.

(2) $\Rightarrow$ (3). By a contrapositive proof. Suppose the associated graph is satisfiable. If there is no constant comparand in S_i, the same assignments to the variables as to their corresponding vertices satisfy S_i. Otherwise there is a 0 vertex in the graph. Assigning $\bar{x}-\bar{0}$ to the variable x satisfies S_i.

(3) $\Rightarrow$ (4). We use induction to prove the contrapositive assertion. It is clear that a graph with one vertex and no positive cycles (equivalently, no positive edges) can be satisfied by any integer whatsoever. For the inductive step, we employ the "elimination of variables" technique of Kuhn [56] and King [69]. Thus take a graph G with no positive cycles. Let G have $k>1$ vertices, and choos a vertex w. Let G' be the graph obtained from G by

1) removing w and all edges incident on w,

2) adding, for each pair of edges in G of the form $u \rightarrow w$ of weigh and $w \rightarrow v$ of weight k, where $u \neq w$, $v \neq w$, a new edge $u \rightarrow v$ of weigh $\ell+k$. (The number of edges added = in-degree of w $\times$ out-degre of w, assuming there is no loop on w in G.)

Obviously, since G does not have a positive cycle, neither does G'. By the inductive hypothesis, we can assign integers to G' satisfying the inequalities associated with its edges, including those edges of G not incident on w. It remains to extend this by assigning a value to w so that all edges of G incident on w are satisfied also.

If G has no edge of the type $u\to w$ for $u\neq w$, or of the type $w\to v$ for $v\neq w$, we need only assign $\bar{w}$ sufficiently large, or small, respectively. If G has edges of both types, then the edges added in step 2) in the construction of G' will guarantee that the upper and lower bounds on w defined by the edges incident on w and by the assignments to G' will leave a nonempty interval from which to make an assignment for w. Formally, let

$$m=\max\{\bar{v}+k \mid w\to v \text{ is an edge of weight } k \text{ in } G,\ v\neq w\}$$

$$M=\min\{\bar{u}-\ell \mid u\to w \text{ is an edge of weight } \ell \text{ in } G,\ u\neq w\}.$$

Then $M\geq m$. For otherwise $m>M$, and there exist edges in G

$u\to w$ of weight ℓ, $u\neq w$,

$w\to v$ of weight k, $v\neq w$,

such that $m=\bar{v}+k$, $M=\bar{u}-\ell$. By construction, we know that G' has an edge $u\to v$ of weight $\ell+k$, so $\bar{u}\geq\bar{v}+(\ell+k)$. But $m>M \Rightarrow \bar{v}+k>\bar{u}-\ell \Rightarrow \bar{v}+(\ell+k)>\bar{u}$, which is a contradiction. Thus let $\bar{w}$ be any integer in the nonempty interval $[m,M]$. Then for each edge $u\to w$ of weight ℓ, $\bar{u}-\ell\geq M\geq\bar{w} \Rightarrow \bar{u}\geq\bar{w}+\ell$. Similarly for edges $w\to v$. Hence all edges of G are satisfied, since any loop on w must have nonpositive weight and is trivially satisfied.

(4) $\Rightarrow$ (1). The positive cycle obviously mirrors an A-proof of an assertion of the form $x>x+c$ or $0>c$, where $c\geq 0$. So $x>x$ or $0>0$ follows by transitivity (after applying trivial monotonicity in the first case). But we have $\neg x>x$, $\neg 0>0$ by irreflexity, and hence an A-proof of contradiction.

§4 The Satisfiability Problem

By the reduction of the last section, we have simplified the task of the arithmetic checker to one of deciding the satisfiability of a set of arithmetic relations whose comparands are either constants or monic linear univariate polynomials (x or $x+c$). We can further restrict the problem to one in which all the comparands are monic linear univariate polynomials, by introducing a new variable x_0 and changing constant comparands c to x_0 (if $c=0$) or x_0+c (if $c\neq 0$). This transformation certainly preserves the satisfiability property of the set of arithmetic relations.

As shown in the previous section, we can replace a conjunction of relations other than $\neq$-relations by an equivalent conjunction of relations of the form $x\geq y[+c]$. Clearly each $\neq$-relation can also be put into the form $x\neq y[+c]$. Once again, we formulate the problem in terms of the satisfiability of a weighted directed graph, but in a slightly different manner than in the previous section. As before, the variables are the vertices, and the $\geq$-relations provide the weighted, directed edges. However, we now leave the $\neq$-relations as additional constraints on the integer assignments. In comparison with the previous approach, we see that a graph satisfiability problem with k $\neq$-constraints is in fact equivalent to 2^k problems without such constraints. This exponential proliferation of unconstrained problems leads us to suspect that the constrained proble would be much harder than the unconstrained problem. Theorems 1 anc 2 below reinforce this suspicion.

Consider a weighted, directed graph G whose vertices are numbere from 1 to say, n. The edge weights are (finite) integers, which are augmented by $\{-\infty,+\infty\}$ so that we can talk about general maximum path weights. The resulting algebraic structure is the closed semiring

$$(\mathbb{Z}^*, -\infty, 0, \max, +)$$

where $\mathbb{Z}^* = \mathbb{Z} \cup \{-\infty, +\infty\}$,

max is the semiring addition, corresponding to weight maximization over a set of paths, with identity $-\infty$ for the maximum over an empty set, and

+ is the semiring multiplication, corresponding to path concatenation, with identity 0 for the null (or 0-edge) path.

Formally, we require $-\infty+x=-\infty$ for all x, consistent with our interpretation of concatenation for + and disconnection for $-\infty$. We also extend the <-relation to $\mathbb{Z}^*$ by requiring

$$-\infty < x \qquad \text{for all } x \neq -\infty$$

$$x < -\infty \qquad \text{for all } x \neq +\infty$$

Also $x^* = \begin{cases} 0 & \text{if } x \le 0 \\ +\infty & \text{if } x > 0. \end{cases}$

As in Algorithm 5.5 (computation of costs between vertices) of AHU [74], let C^k_{ij} denote the maximum weight of paths from i to j without passing through (i.e. entering and leaving) vertices numbered higher than k. If there is no such path, $C^k_{ij} = -\infty$; if the weights of such paths are unbounded from the above, $C^k_{ij} = +\infty$; in all other cases C^k_{ij} is finite. Clearly $C^0_{ij} \neq +\infty$ for all i,j. We shall show that AHU Algorithm 5.5 can be modified to discover a positive cycle if there is one, and to compute the C^n_{ij} values (which are necessarily $<+\infty$) otherwise. It will be seen that $+\infty$ need never enter into the computation.

<u>Thm 1</u> (1) G has a positive cycle $\Leftrightarrow C^k_{ii} > 0$ for some i,k.

(2) Suppose $C^k_{ii} \le 0$ for all i, for all $k<k_0$ for some k_0.

Then $C^k_{ij} < +\infty$ for all i,j and all $k \le k_0$.

<u>Proof</u>: (1) is obvious.

(2) We know that $C_{ij}^{0} < +\infty \, \forall i,j$. Then using

$$C_{ij}^{k} = \max(C_{ij}^{k-1} \, , \, C_{ik}^{k-1} + C_{kk}^{k-1*} + C_{kj}^{k-1})$$

and the fact that $C_{kk}^{k-1*} = 0$ for all $k \le k_0$, we can show by induction that $C_{ij}^{k} < +\infty$ for all k, $0 < k \le k_0$.

<u>Corollary</u>: The unconstrained graph satisfiability problem is solvab in $O(n^3)$ time.

<u>Proof</u>: The existence of positive cycles can be detected by the following modified version of AHU [74] Algorithm 5.5, which also computes the maximum path weights C_{ij}^{n} if no positive cycle is found

```
Initialize $A_{ij}$ to $C_{ij}^{0}$ ;
poscylce := false;
for k := 1 to n while ¬poscycle do
    begin for i:=1 to n while ¬poscycle do
        begin for j:=1 to n do
                $B_{ij} := \max(A_{ij} , A_{ik} + A_{kj})$;
              poscycle := $(B_{ii} > 0)$
        end;
        copy $B_{ij}$ to $A_{ij}$ for all i,j
    end;
```

We have assumed that the integer arithmetic works properly for $\mathbb{Z} \cup \{-\infty\}$. Th^{m} 1 guarantees that $+\infty$ will not be encountered, and that the program is correct.

<u>Th^{m} 2</u>: The constrained graph satisfiability problem is NP-hard.

Proof: By a reduction of the k-colorability problem. Given a connected, undirected graph G and an integer k, we construct a weighted, directed graph G' whose vertices are those of G plus a new vertex v_0. For each vertex v of G, introduce edges $v \to v_0$, $v_0 \to v$ of weights 1 and -k respectively for G'. For each edge (u,v) of G, introduce a constraint $u \neq v$. The resulting constrained problem is satisfiable if and only if G is k-colorable. For connected graphs G, the reduction can be done in polynomial time. Since the k-colorability problem for connected graphs is NP-hard, so is the constrained satisfiability problem.

Corollary: The constrained satisfiability problem for strongly connected graphs is NP-hard.

Proof: The proof of the theorem also proves this corollary.

The next observation allows us to break the problem into subproblems defined by the strongly connected components (SCCs) of the graph, and to ignore inter-SCC edges and constraints. The constraints retained, i.e. those which relate pairs of (not necessarily distinct) vertices both inside the same SCC, are called the internal constraints of their respective SCCs.

$Th^m 3$ A directed, weighted graph G with constraints is satisfiable $\Leftrightarrow$ each SCC, together with its internal constraints, is satisfiable.

Proof: "$\Rightarrow$" is obvious.

"$\Leftarrow$" First make satisfying assignments to each SCC separately. Then consider the quotient graph[+] H = G modulo SCCs, i.e. the vertices

[+] Also known as the condensation of G.

of H are the SCCs of G, and for every pair of SCCs U, V of G, there is an edge $U \rightarrow V$ in H if and only if there exist vertices u, v of G such that $u \in U$, $v \in V$, and there is an edge $u \rightarrow v$ in G. Clearly H is a da so we can do a topological sort on it. Now visit the SCCs in the reverse of the topological order. For each SCC, add an appropriate constant to the assignments of all its vertices so as to satisfy all the inter-SCC edges and constraints involving it and some previously visited SCC.

Th^{m}4 The constrained graph satisfiability problem is in NP, and hence is NP-complete.

Proof: For each constraint $x \neq y$, guess $x > y$ or $x < y$ and add edge to graph accordingly. After making guesses for all constraints (in nondeterministic linear time), run the $O(n^3)$ time algorithm of the corollary to Th^{m}1 on the unconstrained augmented graph to determine its satisfiability. The original constrained graph is satisfiable if and only if the augmented graph for some set of guesses for the constraints is satisfiable. Hence the problem is solvable in non-deterministic $O(n^3)$ time. In conjunction with Th^{m}2 this proves NP-completeness.

Corollary: The constrained problem for strongly connected graphs is NP-complete.

Proof: By Th^{m} 4, and the corollary to Th^{m}2.

Thus the constrained satisfiability problem for strongly connect graphs is a member of a class of problems which are solvable in exponential time, but for which no polynomial time algorithm has yet been devised. The exponential time algorithm in this case can be derived from the equivalence of the constrained problem to an exponential number of unconstrained problems each solvable in polynomial time. As shown in Section 3, this equivalence is obtained by replacing each constraint $x \neq y+c$ with $x>y+c \vee x<y+c$, the exponential explosion arising from the binary choice of each disjunction. However, notice that if more than one constraints relate the same pair of variables, say $x \neq y+c_1, \ldots, x \neq y+c_r$, for $r>1$, and $c_1<c_2<\ldots<c_r$, then many of the resulting unconstrained problems are trivially unsatisfiable, namely, those for which the choices for the disjunctions result in $x<y+c_i$ and $x>y+c_j$ for some $i<j$. A more efficient approach makes use of the following equivalence instead:

$$(*) \quad x \neq y+c_1 \wedge \ldots \wedge x \neq y+c_r$$
$$\equiv x<y+c_1 \vee (x>y+c_1 \wedge x<y+c_2) \vee \ldots$$
$$\vee\ x>y+c_r$$

For a problem with $r_1, r_2, \ldots, r_k$ constraints relating k different pairs of vertices, this alternative approach gives rise to $(1+r_1)(1+r_2)\ldots(1+r_k)$ unconstrained problems. It is easy to show that for all positive integers $r_1, \ldots, r_k$, we have

$$(1+r_1)(1+r_2)\ldots(1+r_k) \leq 2^{r_1+\ldots+r_k}$$

with equality holding if and only if $r_1=r_2=\ldots=r_k=1$. Thus there is a uniform, but not necessarily positive, economy, which roughly increases with the difference $(r_1+\ldots+r_k)-k$.

The problem now becomes one of searching a tree of weighted, directed graphs for a satisfiable leaf graph. At the root of this tree is the original graph. The other levels of the tree are defined by the distinct (unordered) pairs of vertices that are constrained by one or more $\neq$-relations. Suppose there are k levels, represented by the constraints:

$$\bigwedge_{1\le i\le k} \bigwedge_{1\le j\le r_i} (x_i \neq y_i + c_{i,j}) \quad .$$

where $r_i>0$ for all i, and $c_{i,j}<c_{i,j+1}$ for all i,j such that $1\le j<r_i$. By equation (*), these k levels of constraints are equivalent to a conjunction of k disjunctions,

$$\bigwedge_{1\le i\le k} \bigvee_{0\le j\le r_i} P_{i,j}$$

where $P_{i,0}$ denotes $x_i<y_i+c_{i,1}$

$P_{i,j}$ denotes $x_i>y_i+c_{i,j} \wedge x_i<y_i+c_{i,j+1}$

for $0<j<r_i$

and P_{i,r_i} denotes $x_i>y_i+c_{i,r_i}$.

In terms of satisfiability, we have the equivalences

$$P_{i,0} \equiv y_i \ge x_i + (-c_{i,1}+1)$$

$$P_{i,j} \equiv x_i \ge y_i + (c_{i,j}+1) \wedge y_i \ge x_i + (-c_{i,j+1}+1)$$

for $0<j<r_i$

and $P_{i,r_i} \equiv x_i \ge y_i + (c_{i,r_i}+1)$.

This shows that each $P_{i,j}$ represents one or two edges. For each d=1, 2,..., or k, we can rewrite the conjunction of the first d disjunctio above as an equivalent disjunctive normal form

$$\bigvee_{0\le j_1\le r_1,\ldots,0\le j_d\le r_d} (P_{1,j_1} \wedge P_{2,j_2} \wedge\ldots\wedge P_{d,j_d})$$

Each disjunct $P_{1,j_1} \wedge \ldots \wedge P_{d,j_d}$ defines a node at depth d of the tree. The graph of this node is obtained by adding the edges represented by the disjunct to the root graph. Its father is the node defined by the disjunct $P_{1,j_1} \wedge \ldots \wedge P_{d-1,j_{d-1}}$ in the disjunctive normal form for the first d-1 levels. Thus the graph of every node other than the root is obtained by adding one or two edges to the graph of its father.

To search this tree for a satisfiable leaf graph, we employ a recursive depth-first traversal algorithm. Since the descendants of a node with an unsatisfiable graph all have unsatisfiable graphs, there is no need to search the subtree rooted at such a node. However, in order to exploit this fact, we need to be able to decide which of the sons of a node with a satisfiable graph also have satisfiable grpahs. Suppose the node is at depth d-1 with graph G_{d-1}. The constraints corresponding to depth d relate vertices x_d and y_d. Let the maximum path weights in G_{d-1} from x_d to y_d and from y_d to x_d be a,b respectively. Since G_{d-1} is satisfiable, by Th^m 3.4, $a+b\leq 0$, or $a\leq -b$, and these weights define lower and upper bounds for the difference x_d-y_d, i.e. $x_d-y_d \in [a,-b]$. Similarly, $P_{d,j}$ defines bounds for x_d-y_d:

$$P_{d,0} \equiv x_d-y_d \in (-\infty, c_{d,1})$$

$$P_{d,j} \equiv x_d-y_d \in (c_{d,j}, c_{d,j+1}) \quad 0<j<r_d \quad .$$

$$P_{d,r_d} \equiv x_d-y_d \in (c_{d,r_d}, +\infty)$$

The sons of the node have graphs obtained by adding the edges representing these intervals to G_{d-1}. Obviously, if the interval defined by $P_{d,j}$ is disjoint from $[a,-b]$, then the graph of the corresponding son is unsatisfiable. This happens if either the interval is itself empty, i.e. $0<j<r_d$ and $c_{d,j}=c_{d,j+1}-1$, or if it is entirely to the left or right of $[a,-b]$. On the other hand, we shall prove that if this inte is not disjoint from $[a,-b]$, then the graph of the corresponding son is satisfiable, and hence a candidate for recursive search. Along wi the proof, we shall exhibit an $(O(n^2))$ algorithm to update the maximum path weight matrix of a graph to reflect the addition of a new edge. First we present some notation.

We shall be concerned with a strongly connected, directed graph G whose edges are weighted by the integers. A simple path in G is a path which enters and leaves any vertex at most once, so that a cycle is a closed simple path. For a path p in G, we denote its weight by w_p. Also let m_{xy} denote the maximum path weight from vertex x to vertex y, i.e. $m_{xy}=C^n_{xy}$ where n is the number of vertices in G.

Lemma 1: Suppose G has no positive cycle. Then for any path p from vertex x to vertex y, there exists a simple path q from x to y such that $w_p \le w_q$.

Proof: By induction on the length of path p, using the fact that removal of any cycle from a path does not decrease the weight.

Lemma 2: $m_{xy}<+\infty$ for all vertices x,y in G $\Leftrightarrow$ G has no positive cycle.

Proof: "$\Leftarrow$" By lemma 1, all paths from x to y have weights dominated by the weights of the finitely many simple paths among them.

"$\Rightarrow$" Assume G has a positive cycle p through vertex x. Then $m_{xx}=+\infty$ by concatenating p to itself arbitrarily many times.

<u>Lemma 3</u>: If G has no positive cycle, then for any pair of vertices x,y, there is a simple path from x to y such that $w_p = m_{xy}$.

<u>Proof</u>: By lemma 2, $m_{xy} < +\infty$, so there exists a path q from x to y such that $w_q = m_{xy}$. By lemma 1, there is a simple path p from x to y such that $w_p \geq w_q = m_{xy}$. But by definition, $w_p \leq m_{xy}$ also.

<u>Lemma 4</u>: Suppose G has no positive cycle. Let x,y be vertices in G, let edge e from x to y of weight w_e be an edge not in G, and let G' be the graph obtained by adding e to G. Then

$$G' \text{ has a positive cycle } \Leftrightarrow w_e + m_{yx} > 0.$$

where m_{uv} denotes the maximum path weights in G.

<u>Proof</u>: " $\Leftarrow$ " is obvious.

" $\Rightarrow$ " Assume G' has a positive cycle p. If p does not contain e at all, then G has a positive cycle, contrary to assumption. Hence p contains e, but only once since p is simple. Thus p can be decomposed into a path q from y to x that lies completely in G, and the edge e, with

$$w_p = w_e + w_q > 0.$$

But q is a path in G $\Rightarrow$ $w_q \leq m_{yx}$

$\therefore$ $w_e + m_{yx} \geq w_e + w_q > 0$

<u>Th^{m}5</u> Let G, e, G', m_{uv} be as in lemma 4. Let m'_{uv} denote maximum path weights in G'. Suppose G' has no positive cycle. Then for all vertices u,v,

$$m'_{uv} = \max(m_{ux} + w_e + m_{yv}, m_{uv}).$$

Proof: Since m_{uv}, $m_{ux}+w_e+m_{yv}$ are both weights of paths from u to v in G', they are both dominated by m'_{uv}. Hence $m'_{uv} \geq \max(m_{ux}+w_e+m_{yv}, m$ On the other hand, since G' has no positive cycle, by lemma 3 there is a simple path p from u to v such that $m'_{uv}=w_p$. If p does not contain e then p is in G $\Rightarrow w_p \leq m_{uv}$. If p contains e, it contains e exactly once because it is simple. So p can be decomposed into a path q from u to x, the edge e from x to y, and a path r from y to v, where all edges of q, r are in G. Hence

$$w_p=w_q+w_e+w_r$$

where $w_q \leq m_{ux}$, $w_r \leq m_{yv}$.

$\therefore$ $w_p \leq m_{ux}+w_e+m_{yv}$.

In either case, $m'_{uv}=w_p \leq \max(m_{ux}+w_e+m_{yv}, m_{uv})$.

Corollary: (Maximum path weight update). Assumptions as in Th^{m}5.

$$m'_{uv} = \max(m'_{ux}+w_e+m_{yv}, m_{uv}) \qquad (1)$$

$$= \max(m_{ux}+w_e+m'_{yv}, m_{uv}) \qquad (2)$$

$$= \max(m'_{ux}+w_e+m'_{yv}, m_{uv}) \qquad (3)$$

Proof: We first prove (3).

By Th^{m}5, $m'_{uv}=\max(m_{ux}+w_e+m_{yv}, m_{uv})$. Also by Th^{m}5, $m_{xy} \leq m'_{xy}$ for al x,y, so

$$m_{ux} \leq m'_{ux} \text{ and } m_{vy} \leq m'_{vy}$$

$\therefore$ $m'_{uv} \leq \max(m'_{ux}+w_e+m'_{vy}, m_{uv})$

However, m_{uv} , $m'_{ux}+w_e+m'_{vy}$ are both weights of paths from u to v in G', so by definition

$$m'_{uv} \geq \max(m'_{ux}+w_e+m'_{vy} , m_{uv}) \text{ also.}$$

(1) and (2) are proved similarly.

Th^{m}5 gives us an $O(n^2)$ algorithm for updating a maximum path weight matrix; its corollary allows us to perform the update in situ. However, we only need to do the update when the new graph has no positive cycle. The next result allows us to decide when this condition holds.

Th^{m}6 Suppose G has no positive cycle. Let x,y be distinct vertices such that $m_{xy}=a$, $m_{yx}=b$. We know that $a+b\leq 0$, or $a\leq -b$, and that G is satisfiable (by Th^m 3.4). Then for any c such that $a\leq c\leq -b$, there is an assignment which satisfies G and the additional relation $x=y+c$.

Proof: It suffices to show that the graph G", obtained by adding edges p from x to y of weight c and q from y to x of weight $-c$ to G, has no positive cycle. First add p to G to obtain G'. Let m_{uv} , m'_{uv} , m''_{uv} denote the maximum path weights in G, G', G" respectively. Now G has no positive cycle, and $c+m_{yx}=c+b\leq 0$ since $c\leq -b$, so we know that G' has no positive cycle by lemma 4. Then Th^{m}5 is applicable, and

$$\begin{aligned} m'_{xy} &= \max(m_{xx}+c+m_{yy} , m_{xy}) \\ &= \max(0+c+0,a)=c. \end{aligned}$$

Next add q to G' to obtain G" Again G' has no positive cycle, and

$$(-c)+m'_{xy}=-c+c=0 \Rightarrow G" \text{ has no positive cycle.}$$

∴ By Th^m 3.4, G" is satisfiable.

Th^m 6 completes the justification of our criterion for deciding the satisfiability of the graph of a son from the maximum path weights of the father's graph and the edges added for the son. We now state the recursive traversal algorithm in pidgin Algol:

```
boolean procedure nodesat (maxwt,d);
integer array maxwt(n,n); integer d;
comment maxwt is max path weight matrix of father,
        d is depth of sons,
        n = no. of vertices in graph, a global parameter,
        ℓ = no. of levels of constraints;
begin integer array newmaxwt(n,n); integer a,b;
      nodesat := false;
      let x,y be vertices related by level d constraints;
      let level d constraints be $x \neq y+c_1,\ldots,x \neq y+c_r$, $c_1<\ldots<c_r$;
      a := maxwt(x,y); b := maxwt(y,x);
      for each of the intervals $(-\infty,c_1)$, $(c_1,c_2),\ldots,(c_r,+\infty)$
           while ¬nodesat do
          if interval not disjoint from [a,-b] then do
               if d=ℓ-1 then nodesat := true
               else do compute newmaxwt from maxwt and from edges
                               defining the interval, using Thm 5 and
                               its corollary;
                               nodesat := nodesat(newmaxwt, d+1)
                    od
               od
          od
end nodesat;
```

In connection with the computation of newmaxwt, note that for the first edge defining an interval, Th^m 5 suffices since we do not disturb the original maximum path weights in maxwt, but for the second edge (if interval is finite) we have to use the corollary Th^m 5 so that we can justify updating newmaxwt in situ.

The main program is:

```
Run the algorithm in the corollary to Th^m 1 to compute
max path weights of original graph in array A (setting poscycle);
if ¬poscycle then do
   sort the ≠-relations into k levels;
   poscycle := nodesat(A,1)
   od;
```

§5. Implementation

An arithmetic checker for PLCV 2 has been implemented in PL/I (Optimizing Compiler) using the procedures described above. This checker functions as an independent module of the verifier, which invokes it to decide the validity of arithmetic arguments embedded in correctness proofs. The assertions in these arguments are passed to the checker as strings of encoded operators and operands in prefix form. The checker converts the operands, recursively, into their canonical form representations. We adopt a linked list of terms as our canonical form for a polynomial. Each term consists of a product of variables, represented as an ordered string of the encoded variables, and an associated integer coefficient. The linked list is sorted in the lexicographic order of the product strings of variables of the terms. This implies that the terms are ordered in increasing total degrees; in particular, any nonzero constant term always occurs at the head of the list, so that a canonical form polynomial can be easily separated into the canonical forms of its constant and constant-free parts. In the computer program, these polynomials are constructed f the PL/I based structures and pointers. Operations on polynomials make use of standard merge-add techniques.

After all hypotheses (including those obtained by applications of nontrivial monotonicity) and conclusions have been set up in canonical form, we negate the conclusions, and substitute new distinc variables for distinct constant-free polynomials as justified by Th^m 3.3. The procedures of Section 4 are then applied to decide the satisfiability of the system. We summarize the steps involved:

(1) Construct directed, weighted graph from the relations other than $\neq$-relations. The graph is represented as a matrix in which eac entry contains the weight of the edge directed from the vertex of th row to the vertex of the column. An entry corresponding to an actua edge of the graph is <u>significant</u>; all other entries are <u>insignifican</u> and set to $-\infty$. The significant entries of each row, equivalently th edges emanating from the corresponding vertex, are linked into a lis to facilitate graph manipulations.

(2) The graph is separated into its SCCs by a depth first search.

(3) For each SCC, we compute the maximum path weight matrix, and then apply the nodesat algorithm of the last section to this matrix and the sorted internal $\neq$-constraints. As soon as an SCC is found to be unsatisfiable, we terminate with an affirmative answer; otherwise the system is satisfiable, and the original argument is not justifiable by the arithmetic rule.

We conclude this description with the following remarks. First of all, a relation involving the same variable on both sides is equivalent to a relation with constant comparands, and hence either always true or always false. In the former case, the relation is redundant; in the latter case, the system is trivially contradictory.

Secondly, the complexity of the decision problem clearly increases with the dimension, i.e. the number of distinct variables, of the system of inequalities. It is therefore to our advantage to reduce the dimension whenever possible. Such an opportunity arises when an equality relating two different variables is detected. Obviously, one variable can be expressed in terms of the other throughout the system, thereby decreasing the dimension by 1. Since such a substitution might give rise to more equalities, in fact we group variables related by equalities into equivalence classes using the union-find algorithm (AHU[74], Algorithm 4.3) with weighted edges. When all equalities have been processed, we pick a representative from each equivalence class, and substitute it for each non-representative of the same class with an appropriate constant displacement. Note that an equality can be implied by a pair of inequalities (e.g. $x \le y+c$ and $y \le x-c$). The checker looks for both explicit and implicit equalities. These substitutions are allowable because A-proofs can make use of comparand substitution and trivial monotonicity.

Following are some examples of the use of the arithmetic rule.

Example 1

$$\frac{H: \left\{\begin{array}{l} \ x-1 \le y \ \& \ y \le x+1 \\ \&\ x-1 \le z \ \& \ z \le x+1 \\ \&\ y-1 \le z \ \& \ z \le y+1 \end{array}\right.}{C: \ x=y \vee y=z \vee z=x}$$

Intuitively, if three integers x,y,z differ from each other by at most 1, then they cannot be all distinct. This inference is justifiable by the arithmetic rule, and we sketch an outline of an A-proof of contradiction from $H \wedge \neg C$:

(1)	$x-1\ y\ \&\ y\ x+1$	H
(2)	$y=x-1 \quad y=x \quad y=x+1$	A-provable from (1) (use lemma $x<y \leftrightarrow x+1 \leq y$)
(3)	$y \neq x$	$\neg C$
(4)	$y=x-1 \vee y=x+1$	(2) & (3)
(5)	$z=x-1 \vee z=x+1$	similar proof as for (4)
(6)	$y=z-1 \vee y=z+1$	similar proof as for (4)
(7)	$[y=x-1 \vee y=x+1]$ $\&[z=x-1 \vee z=x+1]$	(4) & (5)
(8)	$[y=x-1\ \&\ z=x-1]$ $\vee[y=x-1\ \&\ z=x+1]$ $\vee[y=x+1\ \&\ z=x-1]$ $\vee[y=x+1\ \&\ z=x+1]$	distributive law, (7)
(9)	$(y=z) \vee (z=y+2) \vee (y=z+2) \vee (y=z)$	A-provable from (8) (use trivial monotonicity & equality)
(10)	$(y=z) \vee (z=y+2) \vee (y=z+2)$	(9)
(11)	$y \neq z$	$\neg C$
(12)	$z=y+2 \vee y=z+2$	(10) & (11)
(13)	$[z=y+2 \vee y=z+2]$ $\&[y=z-1 \vee y=z+1]$	(6) & (12)
(14)	$[z=y+2\ \&\ y=z-1]$ $\vee[z=y+2\ \&\ y=z+1]$ $\vee[y=z+2\ \&\ y=z-1]$ $\vee[y=z+2\ \&\ y=z+1]$	distributive law, (13)
(15)	$(z=z+1) \vee (z=z+3)$ $\vee(z+2=z-1) \vee (z+2=z+1)$ Contradiction	A-provable from (14) (use trivial monotonicity & equality)

Example 2

$$H: \left\{ \begin{array}{lll} x_1 \geq x_2 - 1 & x_5 \geq x_6 & x_8 \geq x_9 - 1 \\ x_2 \geq x_3 & x_6 \geq x_7 & x_9 \geq x_{10} \\ x_3 \geq x_4 - 1 & x_7 \geq x_5 - 1 & x_{10} \geq x_8 - 1 \\ x_4 \geq x_1 & & \\ x_5 \leq x_2 + 10 & x_7 \leq x_3 + 700 & x_8 \leq x_6 + 52 \\ x_9 \leq x_4 + 33 & x_{10} \leq x_1 + 58 & x_{10} \leq x_7 - 580 \end{array} \right.$$

$$C: \left\{ \begin{array}{l} x_1 = x_2 \vee x_3 = x_4 - 1 \vee x_5 = x_7 \vee x_6 = x_5 \\ \vee\ x_8 + 1 = x_9 \vee x_{10} = x_9 - 1 \end{array} \right.$$

This inference is invalid. To see this, notice that the SCCs are $\{x_1, x_2, x_3, x_4\}$, $\{x_5, x_6, x_7\}$ and $\{x_8, x_9, x_{10}\}$, so that the last 6 hypothesis relations can be ignored. The negated conclusion $\neg C$ gives 6 internal constraints. Clearly, the SCCs and their respective internal constraints can be individually satisfied by

$$x_1=0,\ x_2=1,\ x_3=0,\ x_4=0,$$
$$x_5=1,\ x_6=0,\ x_7=0,$$
$$x_8=0,\ x_9=0,\ x_{10}=0$$

Hence, H & $\neg C$ is satisfied by, for instance,

$$x_1=2000,\ x_2=2001,\ x_3=2000,\ x_4=2000,$$
$$x_5=1001,\ x_6=1000,\ x_7=1000,$$
$$x_8=0,\ x_9=0,\ x_{10}=0.$$

$\therefore\ H \not\vdash_{\Lambda} C.$

Example 3

$$H: \begin{cases} x<x^2 \\ x\neq 0 \end{cases}$$

$$C:\ 2\leq x \vee x<0$$

This is valid and is justifiable by the arithmetic rule because it involves only one application of nontrivial monotonicity - factoring

(1)	$x<x^2$ & $x\neq 0$	
(2)	$1<x \vee 1>x$	factoring, (1)
(3)	$1<x \vee 0\geq x$	(2)
(4)	$x\neq 0$	(1)
(5)	$(1<x \vee 0\geq x)$ & $x\neq 0$	(3) & (4)
(6)	$(1<x$ & $x\neq 0) \vee (0\geq x \wedge x\neq 0)$	Distributive law, (5)
(7)	$1<x \vee 0>x$	
(8)	$2\leq x \vee 0>x$	

Notice that actually $x<x^2 \Rightarrow x\neq 0$, but this fact is not A-provable.

The arithmetic checker was tested (stand-alone) on the above examples. It gave the correct answers, using 0.36 CPU seconds for execution and 150K bytes of storage.

§6. Conclusion

We have taken an arithmetic proof system designed for PLCV 2 and have shown that it is at least possible to verify proofs in this system by implementing a proof-checker for it. The fact that the algorithm used has an exponential time complexity need not bother us unduly, as arithmetic arguments that occur in program verificatic can be expected to remain within reasonable size limits. It is more important to be able to construct proofs easily in the system; this remains to be seen until PLCV 2 becomes fully operational. We mention several extensions that have been considered.

Currently, the checker can only verify the inference of a given conclusion from given hypotheses. Since the truth of these hypotheses must have been established by preceding arguments or by assumption, and since PLCV 2 maintains a table of all such established assertions, it is theoretically possible to omit the specification of the hypotheses. However, unless we are willing to try all possible subsets of the established quantifier-free assertions, we must be able to select, on the basis of the structure of the conclusion and of any nontrivial monotonicity operator, those assertions that could possibly participate in the argument as hypotheses. Unfortunately, the transitivity of the order relation and the combinations and cancellations of terms that occur in polynomial arithmetic render such a backward selection extremely difficult.

As mentioned in Section 2, to apply factoring the user has to present the factors of the comparands to the checker. Since the common factor is specified (in the "non-zero condition"), the checker could perform the factorization by dividing it into the comparands. We have chosen not to implement polynomial division because we do not consider this facility to be worth the additional complexity in code and data representation that it necessitates. Nevertheless, the current version of the checker relaxes the requirements of explicit factorization for a comparand that is either zero or is the common factor, and it should be equally easy to do the same for cases in which the common factor is a constant.

Finally, it is possible, and probably desirable, to extend the proof system by adding new axioms. For instance, consider the proof

$$\frac{x \neq 0}{x^2 > 0}$$

With the current system, this proof requires two invocations of the arithmetic rule, because the multiplications of x by itself for the cases $x<0$ and $x>0$ use different nontrivial monotonicity axioms:

$$x>0 \ \& \ x>0 \Rightarrow x^2>0$$

and $$x<0 \ \& \ x<0 \Rightarrow x^2>0.$$

(Note that in general, $x \neq 0$ and $y \neq 0$ only allow us to conclude $xy \neq 0$). Adding a "squaring axiom" will remove the awkwardness in this case, and other extensions may suggest themselves when the system comes into use.

Acknowledgements. The author is indebted to R. L. Constable, S. D. Johnson, and M. J. O'Donnell for many helpful suggestions.

Bibliography

Aho, A.V., J.E. Hopcroft, and J.D. Ullman (AHU) [74]. The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Massachusetts.

Constable, R.L. and M.J. O'Donnell [78]. A Programming Logic, Winthrop, Cambridge, 1978.

Constable, R.L. and S.D. Johnson [78]. Program Verification Reference Manual with User's Guide to PL/CV 2. Department of Computer Science, Cornell University, 1978.

King, J.C. [69]. "A program verifier", Ph.D. Thesis, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pa.

Kleene, S.C. [52]. Introduction to Metamathematics, D. Van Nostrand Company Inc., Princeton, N.J.

Kuhn, H.W. [56]. "Solvability and consistency for linear equations and inequalities", American Math. Monthly, 63, April 1956.

Appendix Arithmetic Axioms for PLCV 2

(1) Ring axioms and the definition of minus,-.

For all integers x,y,z

(i)	$x+y=y+x$	commutativity
	$x*y=y*x$	
(ii)	$(x+y)+z=x+(y+z)$	associativity
	$(x*y)*z=x*(y*z)$	
(iii)	$x*(y+z)=x*y+x*z$	distributivity
(iv)	$x+0=x$	additive identity
(v)	$x*1=x$	multiplicative identity
(vi)	$x+(-x)=0$	additive inverse
(vii)	$x-y=x+(-y)$	subtraction

(2) Discrete linear order

For all integers x,y,z

(i)	$\neg(x<x)$	irreflexivity
(ii)	$x<y \vee y<x \vee x=y$	trichotomy
(iii)	$x<y \;\&\; y<z \Rightarrow x<z$	transitivity
(iv)	$\neg(x<y<x+1)$	discreteness

(3) Definitions of order relations and inequality

For all integers x,y,z

(i) $x\le y \Leftrightarrow x<y \vee x=y$

(ii) $x>y \Leftrightarrow y<x$

(iii) $x\ge y \Leftrightarrow x>y \vee x=y$

(4) Monotonicity of + and *

For all integers w,x,y,z

(i) $x \geq y$ & $z \geq w \Rightarrow x+z \geq y+w$ monotonicity of +
If z,w are constants, this is called an instance of <u>trivial monotonicity</u>.

(ii) $x \geq y$ & $z \leq w \Rightarrow x-z \geq y-w$ monotonicity of -
If z and w are constants, this is called an instance of <u>trivial monotonicity</u>.

(iii) $x \geq 0$ & $y \geq z \Rightarrow x*y \geq x*z$ monotonicity of *

(iv) $x > 0$ & $x*y > x*z \Rightarrow y > z$ cancellation (factoring)

To make the proof system more powerful, many variants of the above axioms are incorporated into the arithmetic proof rule. These variants, modulo the ring axioms, are given in the following tables, in which each entry contains the conclusion from the hypotheses corresponding to its row and column.

Addition	$z>w$	$z \geq w$	$z=w$	$z \neq w$
$x>y$	$x+z \geq y+w+2$	$x+z \geq y+w+1$	$x+z \geq y+w+1$ & $x+w \geq y+z+1$	X
$x \geq y$	$x+z \geq y+w+1$	$x+z \geq y+w$	$x+z \geq y+w$ & $x+w \geq y+z$	X
$x=y$	$x+z \geq y+w+1$ & $y+z \geq x+w+1$	$x+z \geq y+w$ & $y+z \geq x+w$	$x+z=y+w$ & $x+w=y+z$	$x+z \neq y+w$ & $x+w \neq y+z$
$x \neq y$	X	X	$x+z \neq y+w$ & $x+w \neq y+z$	X

Subtraction

	$z>w$	$z\geq w$	$z=w$	$z\neq w$
$x>y$	$x-w\geq y-z+2$	$x-w\geq y-z+1$	$x-w\geq y-z+1$ & $x-z\geq y-w+1$	X
$x\geq y$	$x-w\geq y-z+1$	$x-w\geq y-z$	$x-w\geq y-z$ & $x-z\geq y-w$	X
$x=y$	$x-w\geq y-z+1$ & $y-w\geq x-z+1$	$x-w\geq y-z$ & $y-w\geq x-z$	$x-w=y-z$ & $y-w=x-z$	$x-w\neq y-z$ & $x-z\neq y-w$
$x\neq y$	X	X	$x-w\neq y-z$ & $x-z\neq y-w$	X

Multiplication

	$y\geq z$	$y>z$	$y=z$	$y\neq z$
$x>0$	$xy\geq xz$	$xy>xz$	$xy=xz$	$xy\neq xz$
$x\geq 0$	$xy\geq xz$	$xy\geq xz$	$xy=xz$	X
$x=0$	$xy=xz$ & $xy=0$	$xy=xz$ & $xy=0$	$xy=xz$ & $xy=0$	$xy=xz$ & $xy=0$
$x\leq 0$	$xy\leq xz$	$xy\leq xz$	$xy=xz$	X
$x<0$	$xy\leq xz$	$xy<xz$	$xy=xz$	$xy\neq xz$
$x\neq 0$	X	$xy\neq xz$	$xy=xz$	$xy\neq xz$

Cancellation (factoring)

	$xy>xz$	$xy\geq xz$	$xy=xz$	$xy\neq xz$
$x>0$	$y>z$	$y\geq z$	$y=z$	$y\neq z$
$x<0$	$y<z$	$y\leq z$	$y=z$	$y\neq z$
$x\neq 0$	$y\neq z$	X	$y=z$	$y\neq z$