On the Simplification and Equivalence Problems for Straight-Line Programs

OSCAR H. IBARRA AND BRIAN S. LEININGER

University of Minnesota, Minneapolis, Minnesota

Abstract The simplification and equivalence problems are examined for several classes of straight-line programs. It is shown that the problems are unsolvable for all nontrivial classes. For example, it is proved that there is no algorithm to determine if an arbitrary program using only the constructs $x \leftarrow 1$, $x \leftarrow x + y$, $x \leftarrow x/y$, where x/y is integer division with truncation, computes the function which has value 1 for all integer inputs. The result holds even if one considers only programs with three input variables which compute total 0/1-functions. Thus, under any criteria of simplification, no algorithm exists for simplifying such programs When $x \leftarrow x + y$ is replaced by $x \leftarrow x - y$, the number of input variables can be reduced to two This is the best possible, since equivalence is decidable for $\{x \leftarrow 1, x \leftarrow x + y, x \leftarrow x - y, x \leftarrow x * y, x \leftarrow x/y\}$-programs with one input variable All the results translate directly to similar results concerning arithmetic expressions.

Categories and Subject Descriptors: D.3.1 **[Programming Languages]**: Formal Definitions and Theory; F.2 1 **[Analysis of Algorithms and Problem Complexity]**: Numerical Algorithms and Problems; F 2 2 **[Analysis of Algorithms and Problem Complexity]**· Nonnumerical Algorithms and Problems; F.3.3 **[Logics and Meanings of Programs]**: Studies of Program Constructs; F.4.1 **[Mathematical Logic and Formal Languages]**. Mathematical Logic

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Equivalence, one-equivalence, simplification, straight-line program, decidability, undecidability, Hilbert's tenth problem

1. *Introduction*

It is well known that the equivalence and simplification problems for programs with loops are unsolvable [13, 14]. (The equivalence problem is: Given two programs, are they defined at the same points and equal wherever they are defined? The simplification problem is: Construct the simplest program, under some cost criteria, equivalent to a given program.) In this paper we consider simple classes of straight-line programs where the only instructions allowed are the arithmetic operations and simple assignment statements. For these classes we can show that the one-equivalence and simplification problems are unsolvable even under "very restricted" use of the arithmetic operations. (The one-equivalence problem is: Given a program, does it compute the one-function, i.e., the function which is identically equal to 1?) We list below our main results. The first three results hold when the inputs range over **Z**, the

This research was supported by the National Science Foundation under Grant MCS 78-01736.

Authors' address. Computer Science Department, 136 Lind Hall, Institute of Technology, University of Minnesota, Minneapolis, MN 55455

set of all integers, or over N, the set of all nonnegative integers. x/y is integer division with truncation (e.g., $5/2 = 2$, $-5/2 = -2$), and $x \div y$ is rational division.

(1) One-equivalence is undecidable[1] for $\{x \leftarrow 1, x \leftarrow x + y, x \leftarrow x/y\}$-programs[2] with at most ten input variables and four auxiliary variables. Hence there is no algorithm for simplifying such programs under any cost criteria. The result holds even if we consider only programs that compute total 0/1-functions (i.e., total functions with range $\{0, 1\}$).[3] When $x \leftarrow x + y$ is replaced by $x \leftarrow x - y$, the numbers of input variables and auxiliary variables can be reduced to two and nine, respectively. This latter result is the best possible, since it is known [8] that equivalence is decidable for $\{x \leftarrow 1, x \leftarrow x + y, x \leftarrow x - y, x \leftarrow x * y, x \leftarrow x/y\}$ -programs with one input variable (but with an unrestricted number of auxiliary variables).

(2) One-equivalence is undecidable for $\{x \leftarrow 1, x \leftarrow x - y, x \leftarrow x * y, x \leftarrow x/2\}$-programs with at most nine input variables and three auxiliary variables. On the other hand, for $\{x \leftarrow 1, x \leftarrow x + y, x \leftarrow x * y, x \leftarrow x/d \mid d \geq 2\}$-programs, one-equivalence is decidable.

(3) Equivalence is decidable for total $\{x \leftarrow 1, x \leftarrow x + y, x \leftarrow x - y, x \leftarrow x * y, x \leftarrow x \div y\}$-programs. When one of the programs is not known to be total, one-equivalence is undecidable. In fact:

(4) One-equivalence is undecidable for $\{x \leftarrow 1, x \leftarrow x + y, x \leftarrow x \div y\}$-programs with at most twenty-eight input variables and three auxiliary variables, when the inputs range over Z. In contrast, when the inputs range over N, equivalence is decidable for $\{x \leftarrow 0, x \leftarrow 1, x \leftarrow x + y, x \leftarrow x * y, x \leftarrow x \div y\}$-programs.

The next two results concern programs whose inputs range over N.

(5) Equivalence of $\{x \leftarrow 1, x \leftarrow y, x \leftarrow x \dot{-} y, x \leftarrow x/y\}$-programs[4] with at most two input variables and six auxiliary variables is undecidable. However, the problem becomes decidable if one of the programs being considered computes a total function.

(6) One-equivalence is undecidable for $\{x \leftarrow 1, x \leftarrow x \dot{-} y, x \leftarrow x * y\}$-programs with at most nine input variables and three auxiliary variables.

The results above should be contrasted with "positive" results (see, e.g., [2–4, 16]) concerning the simplification and equivalence problems for straight-line programs for which no algebraic laws (like distributivity or associativity between operators) hold. For such programs, simplification and equivalence are either NP-hard or polynomial-time solvable [2–4, 16]. (See [1, 5] for the definitions of NP-hard, NP-complete, etc.)

The proofs for the unsolvable problems rely on the following result [11, 12]. (The second reference contains a proof of the result for thirteen variables. The reduction to nine variables, due to Matijasevič, was reported in [11].)

THEOREM 1. *It is undecidable to determine if an arbitrary Diophantine polynomial $F(x_1, \ldots, x_r)$ with at most nine variables has a positive integer solution (i.e., positive integers $a_1, \ldots, a_r$ such that $F(a_1, \ldots, a_r) = 0$).*

[1] We shall often use the term (un) decidable instead of (un) solvable for problems involving yes or no answer

[2] $\{i_1, \ldots, i_r\}$-programs denotes the class of programs using only instruction types $i_1, \ldots, i_r$.

[3] A program is *total* if no division by 0 occurs

[4] $x \dot{-} y$ equals $x - y$ if $x \geq y$, 0 otherwise.

Using the fact that every nonnegative integer can be written as a sum of three triangular numbers [17], we also have[5]

THEOREM 1′. *It is undecidable to determine if an arbitrary Diophantine polynomial $F(x_1, \ldots, x_r)$ with at most twenty-seven variables has an integer solution (i.e., solution in $\mathbf{Z}^r$).*

Throughout we shall make the following (natural) assumptions concerning straight-line programs:

(1) Let $x \leftarrow 1$ be in the instruction set defining a class C of programs. Then the one-instruction program

$$x \leftarrow 1$$

is the simplest program (under any cost criteria) in C computing the one-function. It follows that if the one-equivalence problem is undecidable, then the simplification problem is unsolvable.

(2) If the equivalence problem for a class C of programs is decidable, then there is an algorithm for simplifying programs in C.

(3) The input/output instructions *Input* $(x_1, \ldots, x_n)$ and *Output* $(y_1, \ldots, y_m)$, which appear only at the beginning and end of each program, respectively, are not part of the instruction set.

2. *Straight-Line Programs with Division*

In this section we investigate the one-equivalence, equivalence, and simplification problems for straight-line programs whose inputs range over $\mathbf{Z}$ or $\mathbf{N}$. We study several classes of programs where each class is defined by an instruction set which is a subset of the following types of instructions: $x \leftarrow 0$, $x \leftarrow 1$, $x \leftarrow x + y$, $x \leftarrow x - y$, $x \leftarrow x * y$, $x \leftarrow x/y$ (integer division with truncation), and $x \leftarrow x \div y$ (rational division).

Convention. In this section, when a result is stated without the input domain being specified, the result holds for the cases when the inputs range over $\mathbf{Z}$ or $\mathbf{N}$.

We begin with the following lemma.

LEMMA 1. *Let $g(z)$ be the function*

$$g(z) = \begin{cases} z & \text{if } z \neq -1, \\ 1 & \text{if } z = -1. \end{cases}$$

Then $g(z)$ can be computed by an $\{x \leftarrow 1, x \leftarrow x + y, x \leftarrow x/y\}$-program using only z and two auxiliary variables u and w.

PROOF. Clearly, $g(z) = z/(2(1/(2z + 1)) + 1)$ works, and one can easily write the desired program to compute this function. □

Our first theorem is rather surprising.

THEOREM 2. *The one-equivalence problem for $\{x \leftarrow 1, x \leftarrow x + y, x \leftarrow x/y\}$-programs with at most ten input variables and four auxiliary variables is undecidable. The result holds even if we consider only programs that compute total 0/1-functions.*

[5] A number is triangular if it is equal to $i(i + 1)/2$ for some integer i.

PROOF. Let F be a Diophantine polynomial with $r \le 9$ variables. Let

$$F = F(x_1, \ldots, x_r) = \sum_{j=1}^{m} c_j x_1^{j_1} \cdots x_r^{j_r} - \sum_{j=m+1}^{n} c_j x_1^{j_1} \cdots x_r^{j_r},$$

where $c_j > 0$ and $j_i \ge 0$ for $1 \le j \le n$ and $1 \le i \le r$. By Theorem 1, it is undecidable to determine for such an instance F whether it has a solution, that is, whether there exist positive integers $x_1, \ldots, x_r$ such that $F(x_1, \ldots, x_r) = 0$. We will use this fact to prove our result. In what follows, $K_i = \max\{j_i \mid 1 \le j \le n\}$ for $1 \le i \le r$. We will construct a program P_F with $r + 1$ input variables $x_1, \ldots, x_r, z$ such that P_F computes the one-function if and only if F has no solution. The construction is complicated by the fact that there are no multiplication and subtraction instructions. It is further complicated by our desire to have the result hold for *total* programs.

The program P_F has the following form:

```
Input (x_1, ..., x_r, z)
  Initialization
  α_1
  .
  .
  .
  α_m
  β_{m+1}
  .
  .
  .
  β_n
  Finalization
Output (z)
```

The initialization section sets the input variables to the following values:

$$x_i \leftarrow g(x_i) + 1 \qquad \text{for} \quad 1 \le i \le r$$
$$z \leftarrow g(z)$$

Clearly, $x_i \ne 0$ and $z \ne -1$ after the initialization. We say that the original input $(x_1, \ldots, x_r, z)$ is *valid* if and only if after the initialization section of P_F, $x_i \ge 1$, and $z = x_1^{K_1} \cdots x_r^{K_r} - 1$.

The section of code α_j $(1 \le j \le m)$ computes the jth term of F. This term is computed by setting a variable v equal to $z + 1$. If the input is valid, then $v = x_1^{K_1} \cdots x_r^{K_r}$. We then divide v by $x_1^{K_1 - j_1} \cdots x_r^{K_r - j_r}$, making v equal $x_1^{j_1} \cdots x_r^{j_r}$. Then v is added c_j times to a variable u which keeps the sum of the positive terms of F. The section of code β_j $(m + 1 \le j \le n)$ is identical to α_i except the terms are collected in a variable w. At the end of β_n, the variable u contains the sum of the positive terms of F while w contains the sum of the negative terms. All this is true, of course, only if the input is valid.

It is clear that the finalization portion of P_F must do the following things:

(1) Check that the input is valid.
(2) Check that $u = w$.

For the input to be valid, the following conditions must be satisfied:

(a) $x_i \ge 1$ for $1 \le i \le r$. (Note that $x_i \ne 0$.)
(b) $z = x_1^{K_1} \cdots x_r^{K_r} - 1$.

Let

$$d_1 = \frac{1}{4x_1/(4x_1 + 1) + 1} + \cdots + \frac{1}{4x_r/(4x_r + 1) + 1}.$$

Then $0 \leq d_1 \leq r$, and $d_1 = r$ if and only if $x_i \geq 1$ for $1 \leq i \leq r$. Hence, a code segment to verify that $x_i \geq 1$ for $1 \leq i \leq r$ can be written using the function

$$\frac{d_1}{r} = \begin{cases} 1 & \text{if } x_i \geq 1 \text{ for } 1 \leq i \leq r, \\ 0 & \text{otherwise.} \end{cases}$$

Now let

$$d_2 = \frac{z}{x_1^{K_1} \cdots x_r^{K_r}} \quad \text{and} \quad d_3 = \frac{z+1}{x_1^{K_1} \cdots x_r^{K_r}}.$$

Then $d_2 = 0$ and $d_3 \geq 1$ if and only if $x_1^{K_1} \cdots x_r^{K_r} \geq 1$ and $z = x_1^{K_1} \cdots x_r^{K_r} - 1$. Code segments can be written to verify that $d_2 = 0$ and $d_3 \geq 1$ using the functions

$$\frac{2d_2+2}{4d_2+2} = \begin{cases} 1 & \text{if } d_2 = 0, \\ 0 & \text{if } d_2 \neq 0; \end{cases}$$

$$\frac{2d_3+2}{2/(3d_3+1)+2d_3+1} = \begin{cases} 1 & \text{if } d_3 \geq 1, \\ 0 & \text{if } d_3 \leq 0. \end{cases}$$

To check that $u = w$, let

$$d_4 = \frac{4u+1}{4w+1} \quad \text{and} \quad d_5 = \frac{4w+1}{4u+1}.$$

Then $u = w$ if and only if $d_4 \neq 0$ and $d_5 \neq 0$. Let

$$d_6 = \frac{1}{4d_4+1} + \frac{1}{4d_5+1}.$$

Then $u = w$ if and only if $d_6 = 0$, and a code segment to verify this can be written using the function

$$\frac{2d_6+2}{4d_6+2} = \begin{cases} 1 & \text{if } d_6 = 0, \\ 0 & \text{if } d_6 \neq 0. \end{cases}$$

It follows from the above discussion that an $\{x \leftarrow 1, x \leftarrow x+y, x \leftarrow x/y\}$-program P_F can be written so that it outputs 1 for all inputs if and only if $F(x_1, \ldots, x_r)$ has no solution over the positive integers. Moreover, P_F need only use four auxiliary variables in addition to the $r+1$ input variables. Details can be found in [7]. □

COROLLARY 1. *There is no algorithm for simplifying $\{x \leftarrow 1, x \leftarrow x+y, x \leftarrow x/y\}$-programs with at most ten input variables and four auxiliary variables. The result holds even if we consider only programs that compute total 0/1-functions.*

COROLLARY 2. *It is undecidable to determine if an arbitrary $\{x \leftarrow 1, x \leftarrow x+y, x \leftarrow x/y\}$-program with at most ten input variables and four auxiliary variables computes a total function.*

The number of input variables in Theorem 2 can be reduced to two if we replace the instruction $x \leftarrow x + y$ by $x \leftarrow x - y$. The proof uses the following lemma.

LEMMA 2. *There is a total $\{x \leftarrow 1, x \leftarrow x-y, x \leftarrow x/y\}$-program P with input variables x, s and output variables t, x, s which computes a function h: $\mathbf{Z}^2 \to \mathbf{Z}^3$ with the following properties:*

(1) *For every (x, s) in $\mathbf{Z}^2$, if $h(x, s) = (c, d, e)$, then $c \neq 0$, $d \neq 0$, $e \neq 0$.*
(2) *For every (c, d, e) with $c, d \geq 1$, $e \geq 2$ there exists (x, s) with $x, s \geq 2$ such that $h(x, s) = (c, d, e)$.*

The result also holds when the domain $\mathbf{Z}^2$ is replaced by $\mathbf{N}^2$.

PROOF. Consider the following program:

```
$s \leftarrow g(s) + 1$    $s \neq 0$.
$t \leftarrow x/s$         If $\underbrace{x = cd + cs \text{ and } s = cde,\ c, d \geq 1,\ e \geq 2}_{(*)}$,
                           then $t = c$.
$t \leftarrow t - 1$
$t \leftarrow g(t) + 1$    $t \neq 0$. If $(*)$, then $t = c$.
$x \leftarrow x/t$         If $(*)$, then $x = d + s$.
$x \leftarrow x - s$       If $(*)$, then $x = d$.
$x \leftarrow x - 1$
$x \leftarrow g(x) + 1$    $x \neq 0$. If $(*)$, then $x = d$.
$s \leftarrow s/t$
$s \leftarrow s/x$         If $(*)$, then $s = e$.
$s \leftarrow s - 1$
$s \leftarrow g(s) + 1$    $s \neq 0$. If $(*)$, then $s = e$.
```

The program above can easily be transformed to a program P over the instruction set $\{x \leftarrow 1, x \leftarrow x - y, x \leftarrow x/y\}$. In addition to x, s, t, the program P will need two other variables u and w (for computing g). □

We can now use Lemma 2 to prove the following corollary to Theorem 2.

COROLLARY 3. *The one-equivalence problem for* $\{x \leftarrow 1, x \leftarrow x - y, x \leftarrow x/y\}$*-programs with two input variables and nine auxiliary variables is undecidable. The result holds even if we consider only programs that compute total* 0/1*-functions.*

PROOF. We use the function h of Lemma 2 to generate the values of $z, x_1, \ldots, x_r$ for the program P_F (of Theorem 2) whenever they are needed. The initialization section is not needed. To generate the value of z, we compute $h(x, s) = (c, d, e)$ and set $z \leftarrow c - 1$. To generate the value of x_1, we compute $h(d, e) = (c_1, d_1, e_1)$ and set $x_1 \leftarrow c_1$. Similarly, to generate the value of x_2, we compute $h(d_1, e_1) = (c_2, d_2, e_2)$ and set $x_2 \leftarrow c_2$, etc. So that we can recompute z and the x_i's, we need to preserve the values of x and s. It follows from Theorem 2 and Lemma 2 that the program P_F (of Theorem 2) can be modified to have only two input variables and nine auxiliary variables. □

An interesting question arises at this point: Can we replace $x \leftarrow x - y$ in Corollary 3 by the instruction $x \leftarrow x + y$? We conjecture that this is possible, but we have no proof at this time. However, we have a partial answer:

COROLLARY 4. *The one-equivalence problem for* $\{x \leftarrow 1, x \leftarrow x + y, x \leftarrow x/y\}$*-programs with three input variables and nine auxiliary variables, where the inputs range over* $\mathbf{Z}$*, is undecidable. The result holds even if we consider only programs that compute total* 0/1*-functions.*

PROOF. Modify the program described in Corollary 3 so that now it has three input variables x, s, p. The first instruction is

```
$p \leftarrow g(p) + 1$    $p \neq 0$. It is equal to $-1$
                           if it was $-2$ before.
```

Thus, when $p = -1$, it can be used to simulate subtractions in the code for h; that is, $x \leftarrow x - y$ can be written as $y \leftarrow y/p$; $x \leftarrow x + y$. □

Remark. Note that the proof of Corollary 4 does not hold if the inputs range over $\mathbf{N}$. When the inputs range over N, Theorem 2 is still our best result.

The two input variables in Corollary 3 cannot be reduced to one. In fact, the following result was shown in [8].

THEOREM 3. *The equivalence problem for* $\{x \leftarrow 1, x \leftarrow x + y, x \leftarrow x - y, x \leftarrow x * y, x \leftarrow x/y\}$*-programs with one input variable*[6] *is decidable. Hence there is an algorithm for simplifying such programs.*

The following corollary was also shown in [8].

COROLLARY 5. *It is decidable to determine if an arbitrary* $\{x \leftarrow 1, x \leftarrow x + y, x \leftarrow x - y, x \leftarrow x * y, x \leftarrow x/y\}$*-program with one input variable computes a total function.*

Although Theorem 3 and Corollary 5 are positive results, the decision procedures are not practically feasible. In fact, it is known that the equivalence problem for $\{x \leftarrow 1, x \leftarrow x + y, x \leftarrow x/2\}$-programs with only *one* input variable (which is also the output variable) and *one* auxiliary variable is NP-hard [9]. The inequivalence problem for $\{x \leftarrow 1, x \leftarrow x + y, x \leftarrow x - y, x \leftarrow x/d | d \geq 2\}$-programs is NP-complete [9].

When division is replaced by multiplication, we have the following proposition, whose proof is straightforward.

PROPOSITION 1. *The equivalence problem for* $\{x \leftarrow 1, x \leftarrow x + y, x \leftarrow x - y, x \leftarrow x * y\}$*-programs is decidable. Hence there is an algorithm for simplifying such programs.*

Adding $x \leftarrow x/2$ to the instruction set in Proposition 1 makes the one-equivalence problem undecidable:

THEOREM 4. *The one-equivalence problem for* $\{x \leftarrow 1, x \leftarrow x - y, x \leftarrow x * y, x \leftarrow x/2\}$*-programs with at most nine input variables and three auxiliary variables is undecidable. It follows that there is no algorithm for simplifying such programs.*

PROOF. With division by 2, it is possible to compute the function

$$f(x, y) = \begin{cases} 0 & \text{if } x = y, \\ 1 & \text{otherwise,} \end{cases}$$

which clearly makes one-equivalence undecidable. $f(x, y)$ can be computed using the formula

$$f(x, y) = (x - y)^2 - \frac{2(x - y)^2 - 1}{2}.$$

Again the details can be found in [7]. □

In Theorem 4, $x \leftarrow x - y$ cannot be replaced by $x \leftarrow x + y$:

THEOREM 5. *The one-equivalence problem for* $\{x \leftarrow 1, x \leftarrow x + y, x \leftarrow x * y, x \leftarrow x/d | d \geq 2\}$*-programs is decidable.*

PROOF. Let P be a program with input variables $x_1, \ldots, x_r$. Let D = product of all divisors d appearing in P (possibly with repetition). Then P computes the one-function if and only if for input $(x_1, \ldots, x_r)$, where each $x_i = 2D$, P outputs 1. □

So far, we have only dealt with "integer" division. In the remainder of this section we shall consider "rational" division. We begin with the following proposition, which is easily verified.

[6] The numbers of output and auxiliary variables are unrestricted.

PROPOSITION 2. *The equivalence problem for* $\{x \leftarrow 1,\ x \leftarrow x+y,\ x \leftarrow x-y,\ x \leftarrow x*y,\ x \leftarrow x \div y\}$*-programs that compute* total *functions is decidable. Hence there is an algorithm for simplifying such programs.*

Proposition 2 is not true if one of the two programs being considered is an arbitrary program (i.e., it may or may not be defined for all inputs). We will show that the one-equivalence problem for $\{x \leftarrow 1, x \leftarrow x + y, x \leftarrow x \div y\}$-programs, where the inputs range over $\mathbf{Z}$, is undecidable. In order to prove this result, we need a variation of Theorem 1′.

LEMMA 3. *It is undecidable to determine for an arbitrary Diophantine polynomial* $F(x_1, \ldots, x_r)$ *with* $r \leq 28$ *variables and positive integer coefficients whether there exist (positive or negative) odd integers* $x_1, \ldots, x_r$ *such that* $F(x_1, \ldots, x_r) = -1$.

PROOF. Given a Diophantine polynomial $F_1(x_1, \ldots, x_r) = \sum_{j=1}^{m} c_j x_1^{j_1} \cdots x_r^{j_r} - \sum_{j=m+1}^{n} c_j x_1^{j_1} \cdots x_r^{j_r}$, where $c_j > 0$ and $j_i \geq 0$ for $1 \leq j \leq n$, define a new polynomial $F_2(x_1, \ldots, x_r, y)$ as follows:

$$F_2(x_1, \ldots, x_r, y) = \left(\sum_{j=1}^{m} c_j x_1^{j_1} \cdots x_r^{j_r} + y \sum_{j=m+1}^{n} c_j x_1^{j_1} \cdots x_r^{j_r} \right)^2 + 2(y+1)^2 + y.$$

Then $F_1(x_1, \ldots, x_r) = 0$ if and only if $F_2(x_1, \ldots, x_r, y) = -1$ for some y.

Now consider the set of all polynomials derived from F_2 by replacing some (possibly none) of the variables $x_1, \ldots, x_r$, say, $x_{k_1}, \ldots, x_{k_t}$, by $x_{k_1} + 1, \ldots, x_{k_t} + 1$, respectively. Then F_2 has value -1 for some input values if and only if one of the polynomials in the set has value -1 for some odd input values. The result follows from Theorem 1′. □

THEOREM 6. *The one-equivalence problem for* $\{x \leftarrow 1, x \leftarrow x + y, x \leftarrow x \div y\}$*-programs with at most twenty-eight input variables and three auxiliary variables, where the inputs range over* $\mathbf{Z}$*, is undecidable. Hence there is no algorithm for simplifying such programs.*

PROOF. Let $F(x_1, \ldots, x_r) = \sum_{j=1}^{n} c_j x_1^{j_1} \cdots x_r^{j_r}$ be a polynomial with $r \leq 28$ variables such that $c_j > 0$ and $j_i \geq 0$ for $1 \leq j \leq n$. Construct the program P_F as follows:

```
Input $(x_1, \ldots, x_r)$
  Initialization
  $\alpha_1$
  .
  .
  .
  $\alpha_n$
  Finalization
Output $(z)$
```

Code for initialization: Sets $u \leftarrow 1$ and $x_i \leftarrow 2x_i + 1$ for $1 \leq i \leq r$. (This guarantees that x_i is odd after initialization.)

Code for α_j, $1 \leq j \leq n$: Sets $u \leftarrow u + c_j * (1 \div (1 \div (x_1^{j_1} \cdots x_r^{j_r})))$.

Code for finalization (note that $u = 0$ if and only if after the initialization $(x_1, \ldots, x_r)$ satisfies $F(x_1, \ldots, x_r) = -1$):

$$w \leftarrow 1 \div u$$
$$z \leftarrow (1 \div u) \div w$$

Then z has value 1 if $u \neq 0$; otherwise, z is undefined. The result now follows from Lemma 3. □

COROLLARY 6. *It is undecidable to determine if an arbitrary* $\{x \leftarrow 1, x \leftarrow x + y, x \leftarrow x \div y\}$*-program with at most twenty-eight input variables and three auxiliary variables, where the inputs range over* $\mathbf{Z}$*, computes a total function.*

Remark. We do not know whether Theorem 6 holds when the input variables are allowed to assume rational values. This problem seems very difficult. Note that if one-equivalence for such programs over rational inputs were decidable, then we would have an algorithm (using the construction of Theorem 6) to determine for an arbitrary Diophantine polynomial whether it has a solution over the rationals—thus settling a well-known open problem [6].

The condition that the inputs range over $\mathbf{Z}$ in Theorem 6 is necessary, since we can prove

THEOREM 7. *The equivalence problem for* $\{x \leftarrow 0, x \leftarrow 1, x \leftarrow x + y, x \leftarrow x * y, x \leftarrow x \div y\}$*-programs,*[7] *where the inputs range over* $\mathbf{N}$*, is decidable.*

PROOF. Let P be a program with input variables $x_1, \ldots, x_r$ containing only instructions of the form $x \leftarrow 0$, $x \leftarrow 1$, $x \leftarrow x + y$, $x \leftarrow x * y$, $x \leftarrow x \div y$. Then P has the following property: Let $1 \leq i_1, \ldots, i_r \leq r$ be distinct indices, and $0 \leq k \leq r$.

(1) If P is undefined (i.e., division by 0 occurs) on input $x_{i_1} = \cdots = x_{i_k} = 0$, $x_{i_{k+1}} = \cdots = x_{i_r} = 1$, then P is undefined on every input $x_{i_1} = \cdots = x_{i_k} = 0$, $x_{i_{k+1}} \geq 1, \ldots, x_{i_r} \geq 1$.
(2) If P is defined on input $x_{i_1} = \cdots = x_{i_k} = 0$, $x_{i_{k+1}} = \cdots = x_{i_r} = 1$, then P is defined on every input $x_{i_1} = \cdots = x_{i_k} = 0$, $x_{i_{k+1}} \geq 1, \ldots, x_{i_r} \geq 1$.

In (1) and (2), no variable is set to 0 if $k = 0$.

Thus two programs P_1 and P_2 with input variables $x_1, \ldots, x_r$ are equivalent (over $\mathbf{N}$) if and only if the following conditions are satisfied:

(3) For any distinct indices $1 \leq i_1, \ldots, i_r \leq r$ and $k \geq 0$, P_1 is defined on input $x_{i_1} = \cdots = x_{i_k} = 0$, $x_{i_{k+1}} = \cdots = x_{i_r} = 1$ if and only if P_2 is defined on the same input.
(4) If P_1 and P_2 are defined on input $x_{i_1} = \cdots = x_{i_k} = 0$, $x_{i_{k+1}} = \cdots = x_{i_r} = 1$, then P_1 and P_2 are equivalent on all inputs such that $x_{i_1} = \cdots = x_{i_k} = 0$ and $x_{i_{k+1}} \geq 1, \ldots, x_{i_r} \geq 1$.

Condition (3) is easily verified. By Proposition 2 and (2) above, (4) can also be verified. It follows that equivalence is decidable. □

Theorem 7 does not hold if $x \leftarrow x + y$ is replaced by $x \leftarrow x - y$:

THEOREM 8. *The one-equivalence problem for* $\{x \leftarrow 1, x \leftarrow x - y, x \leftarrow x \div y\}$*-programs with at most nine input variables and three auxiliary variables, where the inputs range over* $\mathbf{N}$*, is undecidable.*

PROOF. We use Theorem 1. Let

$$F = \sum_{j=1}^{m} c_j x_1^{j_1} \cdots x_r^{j_r} - \sum_{j=m+1}^{n} c_j x_1^{j_1} \cdots x_r^{j_r}.$$

[7] Note that since there is no subtraction, $x \leftarrow 0$ is independent of the other instructions

Construct P_F as follows:

```
Input $(x_1, \ldots, x_r)$
  $x_i \leftarrow x_i + 1$ for $1 \le i \le r$
  $u \leftarrow \sum_{j=1}^{m} c_j x_1^{j_1} \cdots x_r^{j_r} - \sum_{j=m+1}^{n} c_j x_1^{j_1} \cdots x_r^{j_r}$      Computed using the fact that
                                                                $x_1^{j_1} \cdots x_r^{j_r} = 1 \div (1 \div (x_1^{j_1} \cdots x_r^{j_r}))$.
                                                                The variables $w$ and $z$ are used as
                                                                auxiliary variables.
  $w \leftarrow 1 \div u$
  $z \leftarrow (1 \div u) \div w$
Output $(z)$
```

Then z has value 1 for all inputs if and only if F has no solution. □

3. *Straight-Line Programs with Proper Subtraction*

In this section we look at straight-line programs that use proper subtraction $x \leftarrow x \dot{-} y$ instead of the regular subtraction $x \leftarrow x - y$. We assume that the inputs are over the nonnegative integers. We find that there are questions that are decidable with regular subtraction that become undecidable with proper subtraction, and vice versa.

We begin with the following lemma, which is easily verified from Theorem 1.

LEMMA 4. *It is undecidable to determine if a Diophantine polynomial $F(x_1, \ldots, x_r)$ with $r \le 9$ variables has a solution such that $x_1, \ldots, x_r \ge 1$ and $x_i > 1$ for at least one i.*

THEOREM 9. *The one-equivalence problem for $\{x \leftarrow 1, x \leftarrow x \dot{-} y, x \leftarrow x * y\}$-programs with at most nine input variables and three auxiliary variables is undecidable. Hence there is no algorithm for simplifying such programs.*

PROOF. Let $F = F(x_1, \ldots, x_r)$ and K_i be as in Theorem 2; that is,

$$F = \sum_{j=1}^{m} c_j x_1^{j_1} \cdots x_r^{j_r} - \sum_{j=m+1}^{n} c_j x_1^{j_1} \cdots x_r^{j_r},$$
$$K_i = \max\{j_i \mid 1 \le j \le n\} \qquad \text{for} \quad 1 \le i \le r.$$

Let $c = \sum_{j=1}^{n} c_j$, and for $1 \le i \le r$ let $t_i = \lceil \log_2 c \rceil + K_i + 1$.

We will construct a program P_F which outputs 1 for all inputs if and only if there exists no solution to F such that $x_1, \ldots, x_r \ge 1$ and $x_i > 1$ for at least one i. The construction described in Theorem 2 does not work, since there is no addition, and therefore we cannot compare the sum of the positive terms to the sum of the negative terms directly. P_F has the following form:

```
Input $(x_1, \ldots, x_r)$
  Initialization
  $\alpha_1$
  .
  .
  .
  $\alpha_m$
  $\beta_{m+1}$
  .
  .
  .
  $\beta_n$
  Finalization
Output $(z)$
```

We say that the input $(x_1, \ldots, x_r)$ is valid if $x_1, \ldots, x_r \geq 1$ and $x_i > 1$ for at least one i.

Code for initialization:

$$u \leftarrow x_1^{t_1} \cdots x_r^{t_r}$$
$$w \leftarrow x_1^{t_1} \cdots x_r^{t_r}$$

If the input is valid then $u, w > \sum_{j=1}^{n} c_j x_1^{j_1} \cdots x_r^{j_r}$ and, therefore,

$$u \dot{-} \sum_{j=1}^{m} c_j x_1^{j_1} \cdots x_r^{j_r} > 0 \qquad \text{and} \qquad w \dot{-} \sum_{j=m+1}^{n} c_j x_1^{j_1} \cdots x_r^{j_r} > 0.$$

Thus, if the input is valid and $F(x_1, \ldots, x_r) = 0$, then

$$u \dot{-} \sum_{j=1}^{m} c_j x_1^{j_1} \cdots x_r^{j_r} = w \dot{-} \sum_{j=m+1}^{n} c_j x_1^{j_1} \cdots x_r^{j_r}.$$

Code for α_j, $1 \leq j \leq m$:

$$u \leftarrow u \dot{-} c_j x_1^{j_1} \cdots x_r^{j_r}$$

Code for β_j, $m + 1 \leq j \leq n$:

$$w \leftarrow w \dot{-} c_j x_1^{j_1} \cdots x_r^{j_r}$$

Code for finalization (the code checks that the input is valid and $u = w$, i.e., $F(x_1, \ldots, x_r) = 0$):

$$z \leftarrow 1 \dot{-} [(((x_1 \cdots x_r \dot{-} 1) \dot{-} (x_1 \cdots x_r \dot{-} 2)) \dot{-} (u \dot{-} w)) \dot{-} (w \dot{-} u)]$$

It is clear that P_F computes the one-function if and only if F has no solution such that $x_1, \ldots, x_r \geq 1$ and $x_i > 1$ for at least one i. The result now follows from Lemma 4. □

Theorem 9 does not hold for programs with only *one* input variable:

THEOREM 10. *The equivalence problem for* $\{x \leftarrow 1, x \leftarrow x + y, x \leftarrow x \dot{-} y, x \leftarrow x * y\}$*-programs with one input variable is decidable. Hence there is an algorithm for simplifying such programs.*

PROOF. Let P be a program with input variable x and output variables $z_1, \ldots, z_m$. Let l be the length (i.e., the number of instructions) of P. Then the value of any variable y at any time is $0 \leq y \leq 2^{2^l} x^{2^l}$. Let $c = 2^{2^l} l$. Clearly, for $1 \leq k \leq 2^l$, if $x > c$, then

$$x^k - 2^{2^l} x^{k-1} - 2^{2^l} x^{k-2} - \cdots - 2^{2^l} x - 2^{2^l} > 0.$$

It follows that for $x > c$ we can effectively derive unique polynomial expressions (in x) for the output variables $z_1, \ldots, z_m$. Therefore two programs P and P' with compatible input/output variables and length at most l are equivalent if and only if they agree on all inputs $x \leq c$ and the polynomial expressions for their output variables for $x > c$ are identical. □

Remarks

(1) We do not know whether Theorem 10 can be extended to programs with *two* input variables. We note, however, that a positive answer would show the solvability of Hilbert's tenth problems for polynomials with two unknowns—thus resolving an open problem in number theory [6].

(2) It is known that the one-equivalence problem for $\{x \leftarrow 1, x \leftarrow x + y, x \leftarrow x \dot{-} y, x \leftarrow y \dot{-} x\}$-programs with only *one* input variable and *one* auxiliary variable is NP-hard [10].

Next we look at $\{x \leftarrow 1, x \leftarrow y, x \leftarrow x \dot{-} y, x \leftarrow x/y\}$-programs.[8] We begin with the following lemma.

LEMMA 5. *Any $\{x \leftarrow 1, x \leftarrow y, x \leftarrow x \dot{-} y\}$-program has the property that the final value of any variable y satisfies exactly one of the following conditions*:

(1) *y has value 0 for all inputs.*
(2) *y has value 1 for all inputs.*
(3) *y takes on 0 and positive values.*

PROOF. Induction on the length of P. □

LEMMA 6. *Suppose P is an $\{x \leftarrow 1, x \leftarrow y, x \leftarrow x \dot{-} y, x \leftarrow x/y\}$-program computing a total function. Then P is equivalent to a program P' derived from P by deleting instructions of the form $x \leftarrow x/y$.*

PROOF. Consider the first occurrence of an instruction of the form $x \leftarrow x/y$. Then the value of y immediately preceding this instruction satisfies (1), (2), or (3) of Lemma 5. Since P computes a total function, (1) and (3) are not possible. Thus the instruction $x \leftarrow x/y$ is equivalent to $x \leftarrow x$ and can be deleted. It follows that all instructions of the form $x \leftarrow x/y$ can be deleted from P without changing the computed function. □

In [9] it is shown that the equivalence problem for $\{x \leftarrow 1, x \leftarrow y, x \leftarrow x + y, x \leftarrow x \dot{-} y\}$-programs is decidable. Hence, from Lemmas 5 and 6 we have

THEOREM 11. *The equivalence problem for $\{x \leftarrow 1, x \leftarrow y, x \leftarrow x \dot{-} y, x \leftarrow x/y\}$-programs that compute total functions is decidable. Hence there is an algorithm for simplifying such programs.*

COROLLARY 7. *It is decidable to determine if an arbitrary $\{x \leftarrow 1, x \leftarrow y, x \leftarrow x \dot{-} y, x \leftarrow x/y\}$-program P computes a total function.*

PROOF. If P does not have an instruction of the form $x \leftarrow x/y$, it computes a total function. Suppose now that P has an instruction of the form $x \leftarrow x/y$. Consider the first occurrence of such an instruction. Then before the division the value of y is defined for all inputs. By Theorem 11, we can effectively determine which of the three cases of Lemma 5 the value of y satisfies. If y satisfies (1) or (3), then the instruction $x \leftarrow x/y$ cannot be deleted, and P does not compute a total function. If y satisfies (2), we can delete the instruction $x \leftarrow x/y$ being considered. If we are able to delete all instructions of the form $x \leftarrow x/y$, then P computes a total function. □

COROLLARY 8. *Equivalence of two $\{x \leftarrow 1, x \leftarrow y, x \leftarrow x \dot{-} y, x \leftarrow x/y\}$-programs, one of which computes a total function, is decidable.*

PROOF. Direct from Theorem 11 and Corollary 7. □

Corollary 8 becomes false if we remove the requirement that at least one of the programs computes a total function.

THEOREM 12. *The equivalence problem for $\{x \leftarrow 1, x \leftarrow y, x \leftarrow x \dot{-} y, x \leftarrow x/y\}$-programs with at most ten input variables and three auxiliary variables is undecidable.*

[8] Note that since there is no addition, $x \leftarrow y$ is independent of the other instructions.

PROOF. Let F, K_i, and t_i be as in Theorem 9. We will construct two programs P and P_F with input variables $x_1, \ldots, x_r, z$ and output variable u such that P and P_F are equivalent if and only if there are no integers $x_1, \ldots, x_r$ with $x_1 \cdots x_r \geq 2$ such that $F(x_1, \ldots, x_r) = 0$. The program P outputs 1 if $x_i > 0$ for $1 \leq i \leq r$ and $z > 0$. If $x_i = 0$ for some i or $z = 0$, then P is undefined. P is given by the following code:

```
Input $(x_1, \ldots, x_r, z)$
  $u \leftarrow 1/x_i$ for $1 \leq i \leq r$
  $u \leftarrow 1/z$
  $u \leftarrow 1$
Output $(u)$
```

We say that the input $(x_1, \ldots, x_r, z)$ is valid if $z = x_1^{t_1} \cdots x_r^{t_r}$ and $z \geq 2$. (See Theorem 9 for the definition of t_i.) The program P_F computes $F(x_1, \ldots, x_r)$ and outputs 1 if $F(x_1, \ldots, x_r) \neq 0$ and 0 if $F(x_1, \ldots, x_r) = 0$, provided the input is valid. If the input is not valid, then P_F behaves like P. It follows that P is equivalent to P_F if and only if there are no integers $x_1, \ldots, x_r$ with $x_1 \cdots x_r \geq 2$ such that $F(x_1, \ldots, x_r) = 0$. Now

$$F = \sum_{j=1}^{m} c_j x_1^{j_1} \cdots x_r^{j_r} - \sum_{j=m+1}^{n} c_j x_1^{j_1} \cdots x_r^{j_r}.$$

Let

$$A = \sum_{j=1}^{m} c_j x_1^{j_1} \cdots x_r^{j_r} \qquad \text{and} \qquad B = \sum_{j=m+1}^{n} c_j x_1^{j_1} \cdots x_r^{j_r}.$$

Then

$$|F(x_1, \ldots, x_r)| = (A \dot{-} B) + (B \dot{-} A).$$

If the input $(x_1, \ldots, x_r, z)$ is valid, then

$$u = z \dot{-} ((z \dot{-} (A \dot{-} B)) \dot{-} (B \dot{-} A)) = (A \dot{-} B) + (B \dot{-} A).$$

The program P_F computes A, B, and u and outputs 0 if $u = 0$ and the input is valid. P_F has the following form:

```
Input $(x_1, \ldots, x_r, z)$
  Initialization
  $\alpha_1$
  .
  .
  .
  $\alpha_m$
  $\beta_{m+1}$
  .
  .
  .
  $\beta_n$
  Finalization
Output $(u)$
```

Code for initialization:

```
$u \leftarrow 1/x_i$    for  $1 \leq i \leq r$
$u \leftarrow 1/z$
$u \leftarrow z$
$v \leftarrow z$
```

Code for α_j, $1 \leq j \leq m$:

$$w \leftarrow z/x_1^{t_1-j_1} \cdots x_r^{t_r-j_r}$$
$$u \leftarrow u \dot{-} c_j w$$

Code for β_j, $m+1 \leq j \leq n$:

$$w \leftarrow z/x_1^{t_1-j_1} \cdots x_r^{t_r-j_r}$$
$$v \leftarrow v \dot{-} c_j w$$

Clearly, at the end of β_n, if the input $(x_1, \ldots, x_r, z)$ is valid, then

$$u = z \dot{-} \sum_{j=1}^{m} c_j x_1^{j_1} \cdots x_r^{j_r} = z \dot{-} A$$

and

$$v = z \dot{-} \sum_{j=m+1}^{n} c_j x_1^{j_1} \cdots x_r^{j_r} = z \dot{-} B.$$

Code for finalization:

$$\begin{array}{ll} x \leftarrow z \dot{-} u & x = A \\ y \leftarrow z \dot{-} v & y = B \\ u \leftarrow z \dot{-} ((z \dot{-} (x \dot{-} y)) \dot{-} (y \dot{-} x)) & u = (A \dot{-} B) + (B \dot{-} A) \end{array}$$

(Next, we check that the input is valid. Observe that $z = x_1^{t_1} \cdots x_r^{t_r}$ if and only if $(z \dot{-} 1)/x_1^{t_1} \cdots x_r^{t_r} = 0$ and $z/x_1^{t_1} \cdots x_r^{t_r} > 0$.)

$$v \leftarrow (z \dot{-} 1)/x_1^{t_1} \cdots x_r^{t_r}$$
$$w \leftarrow 1 \dot{-} (z/x_1^{t_1} \cdots x_r^{t_r}$$

$$y \leftarrow 1 \dot{-} (z \dot{-} 1) \qquad y = \begin{cases} 0 & \text{if } z \geq 2, \\ 1 & \text{otherwise.} \end{cases}$$

(Note that $F(x_1, \ldots, x_r) = 0$ and the input is valid if and only if $u = v = w = y = 0$.)

$$u \leftarrow 1 \dot{-} ((((1 \dot{-} u) \dot{-} v) \dot{-} w) \dot{-} y)$$

The programs P and P_F have at most $r + 1 \leq 10$ input variables. Moreover, P and P_F can be written using only the constructs $x \leftarrow 1$, $x \leftarrow y$, $x \leftarrow x \dot{-} y$, $x \leftarrow x/y$ with three additional variables. The details of the coding can be found in [7]. □

Theorem 12 can be strengthened:

Corollary 9. *The equivalence problem for* $\{x \leftarrow 1, x \leftarrow y, x \leftarrow x \dot{-} y, x \leftarrow x/y\}$*-programs with two input variables and six auxiliary variables is undecidable.*

Proof. The technique for reducing the number of variables is similar to that used in the proof of Corollary 3. The program to compute $h(x, s)$, which need not be total, is now defined as follows:

```
t ← x
t ← t/s
x ← x/t
x ← x ∸ s
s ← s/t
s ← s/x
```

If at the start of the program, $x = cd + cs$, $s = cde$, $c, d \geq 1$, and $e \geq 2$, then at the end of the program, $t = c$, $x = d$, and $s = e$. The programs P and P_F in Theorem 12 can be modified so that the values of the variables $z, x_1, \ldots, x_r$ are computed from x and s using the function $h(x, s)$ whenever they are needed (see Corollary 3). Details can be found in [7]. □

If we replace instruction $x \leftarrow 1$ by $x \leftarrow x + 1$, we get results which contrast with Theorem 11 and Corollary 7.

THEOREM 13. *The one-equivalence problem for* $\{x \leftarrow x + 1, x \leftarrow y, x \leftarrow x \dot{-} y, x \leftarrow x/y\}$*-programs with at most two input variables and seven auxiliary variables is undecidable. The result holds even if we consider only programs that compute total* $0/1$*-functions. It follows that there is no algorithm for simplifying such programs.*

PROOF. In Theorem 12, change the code for initialization of P_F by $u \leftarrow z$; $v \leftarrow z$, where $z, x_1, \ldots, x_r$ are computed using the following program for the function $h(x, s)$:

$$\begin{array}{ll}
p \leftarrow 1 & \\
t \leftarrow x & \\
\left.\begin{array}{l} s \leftarrow s \dot{-} p \\ s \leftarrow s + 1 \end{array}\right\} & \text{ensures that } s \neq 0 \\
t \leftarrow t/s & \\
\left.\begin{array}{l} t \leftarrow t \dot{-} p \\ t \leftarrow t + 1 \end{array}\right\} & \text{ensures that } t \neq 0 \\
x \leftarrow x/t & \\
x \leftarrow x \dot{-} s & \\
\left.\begin{array}{l} x \leftarrow x \dot{-} p \\ x \leftarrow x + 1 \end{array}\right\} & \text{ensures that } x \neq 0 \\
s \leftarrow s/t & \\
s \leftarrow s/x & \\
\left.\begin{array}{l} s \leftarrow s \dot{-} p \\ s \leftarrow s + 1 \end{array}\right\} & \text{ensures that } s \neq 0
\end{array}$$

□

COROLLARY 10. *It is undecidable to determine if an arbitrary* $\{x \leftarrow x + 1, x \leftarrow y, x \leftarrow x \dot{-} y, x \leftarrow x/y\}$*-program with at most two input variables and seven auxiliary variables computes a total function.*

PROOF. Modify the program P_F of Theorem 12 as described in Theorem 13 and add the following instruction at the end of the program:

$$u \leftarrow u/u$$

□

4. *Conclusions*

We have shown that the problems of simplification and (one-) equivalence for several classes of straight-line programs using only arithmetic operations are unsolvable. The results obviously carry over to more general classes of programs (e.g., programs that allow exponentiation, loops, etc.). All our results can be translated directly to similar results concerning arithmetic expressions. For example, it follows from Corollary 3 that one-equivalence is undecidable for (total) arithmetic expressions whose operands are 1, x, and y and whose operators are minus and integer division. These results should be contrasted with those in [15], which gives some unsolvable problems involving functions of a *real* variable. For example, in [15] it is shown that equivalence is undecidable for expressions whose operands are the variable x, the rational numbers, and the constants π and $\log 2$. The operations are $x + y$, $x - y$, $x*y$, e^x, $\sin x$, and $|x|$.

ACKNOWLEDGMENT. We would like to thank the referees for suggestions which tightened some of our results.

REFERENCES

1. AHO, A.V., HOPCROFT, J.E., AND ULLMAN, J.D. *Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass., 1974.
2. AHO, A V., AND JOHNSON, S.C Optimal code generation for expression trees. In *Proc. 7th Ann. ACM Symp. on Theory of Computing* (Albuquerque, N.M., May 1975), ACM, New York, pp. 207–217.
3. AHO, A.V., AND ULLMAN, J.D. Optimization of straight line programs. *SIAM J. Comput 1*, 1 (Mar 1972), 1–19
4. BRUNO, J, AND SETHI, R. Code generation for a one-register machine. *J. ACM 23*, 3 (July 1976), 502–510.
5. COOK, S A. The complexity of theorem-proving procedures. In *Proc. 3rd Ann. ACM Symp. on Theory of Computing* (Shaker Heights, Ohio, May 1971), ACM, New York, pp. 151–158.
6. DAVIS, M., MATIJASEVIČ, Y, AND ROBINSON, J. Hilbert's tenth problem. Diophantine equations: Positive aspects of a negative solution. *Proc Symp. on Pure Mathematics 28* (1976), 323–378.
7. IBARRA, O H., AND LEININGER, B.S. On the simplification and equivalence problems for straight-line programs. Tech Rep 79-21, Dep. of Computer Science, Univ of Minnesota, Minneapolis, Minn, Sept 1979.
8. IBARRA, O.H., AND LEININGER, B S. Straight-line programs with one input variable. *SIAM J. Comput. 11* (1982), 1–14.
9. IBARRA, O.H., AND LEININGER, B.S. The complexity of the equivalence problem for simple loop-free programs. *SIAM J Comput. 11* (1982), 15–27.
10. IBARRA, O H., AND LEININGER, B.S. On the zero-inequivalence problem for loop programs. *J. Comput Syst Sci.* (to appear).
11. MANDERS, K., AND ADLEMAN, L. NP-complete decision problems for quadratic polynomials In *Proc. 8th Ann. ACM Symp. on Theory of Computing* (Hershey, Pa., May 1976), ACM, New York, pp. 23–29.
12. MATIJASEVIČ, Y., AND ROBINSON, J Reduction of an arbitrary Diophantine equation to one in 13 unknowns. *Acta Arith. 27* (1975), 521–553
13. MEYER, A., AND RITCHIE, D. The complexity of loop programs In *Proc. 22nd National Conf of the ACM*, Thompson Book Co., Washington, D.C., 1967, pp 465–469.
14. MINSKY, M. *Computation: Finite and Infinite Machines.* Prentice-Hall, Englewood Cliffs, N J., 1967
15. RICHARDSON, D. Some undecidable problems involving elementary functions of a real variable. *J. Symb Logic 33*, 4 (Dec 1968), 514–520.
16. SETHI, R., AND ULLMAN, J D The generation of optimal code for arithmetic expressions *J. ACM 17*, 4 (Oct. 1970), 715–728
17. STEWART, B M. *Theory of Numbers*, 2nd ed. Macmillan, New York, 1964.

RECEIVED NOVEMBER 1979; REVISED JANUARY 1981; ACCEPTED JULY 1982