SIAM J. COMPUT.
Vol. 11, No. 1, February 1982

THE COMPLEXITY OF THE EQUIVALENCE PROBLEM FOR SIMPLE LOOP-FREE PROGRAMS*

OSCAR H. IBARRA† AND BRIAN S. LEININGER†

Abstract. We consider a simple class of loop-free programs whose instruction repertoire consists of $x \leftarrow 0$, $x \leftarrow c$, $x \leftarrow cx$, $x \leftarrow x/c$, $x \leftarrow x+y$, $x \leftarrow x-y$, **skip** l, **if** $p(x, y)$ **then skip** l, and **halt**. (x and y are integer variables, c is a positive integer, x/c is integer division, l is a nonnegative integer, and $p(x, y)$ is a predicate of the form $x > y$, $x \geqq y$, $x = y$, $x \neq y$, $x \leqq y$, or $x < y$; **skip** l causes the $(l+1)$st instruction following the current instruction to be executed next.) We show that the equivalence problem for this class is decidable in $2^{\lambda N^2}$ time (N = sum of the sizes of the programs and λ is a fixed positive constant). The bound cannot be reduced to a polynomial in N unless P = NP. In fact, we have the following rather surprising result: The equivalence problem for programs with one input variable (which also serves as the output variable) and one auxiliary variable using only instructions $x \leftarrow 2x$, $x \leftarrow x/2$, and $x \leftarrow x+y$ is NP-hard.

Key words. Complexity, equivalence, zero-equivalence, loop-free programs, NP-hard

1. Introduction.

In an earlier paper [6], we showed that the equivalence problem[1] for several classes of straight-line programs (over positive, negative, or zero integer inputs) using only arithmetic operations is undecidable. In particular, we showed the following:

(a) The one-equivalence problem[2] is undecidable for $\{x \leftarrow 1, x \leftarrow 2x, x \leftarrow x+y, x \leftarrow x/y\}$-programs[3].

(b) The one-equivalence problem is undecidable for $\{x \leftarrow 1, x \leftarrow x/2, x \leftarrow x-y, x \leftarrow x^*y\}$-programs.

In this paper, we study a simple class of straight-line programs with a decidable equivalence problem. Specifically, we consider the class of loop-free programs whose instruction repertoire is $R = \{x \leftarrow 0, x \leftarrow c, x \leftarrow cx, x \leftarrow x/c, x \leftarrow x+y, x \leftarrow x-y$, **skip** l, **if** $p(x, y)$ **then skip** l, **halt**$\}$. x and y are distinct integer variables, c is any positive integer, l is any nonnegative integer, and $p(x, y)$ is a predicate of the form $x > y$, $x \geqq y$, $x = y$, $x \neq y$, $x \leqq y$, or $x < y$. **skip** l causes the $(l+1)$st instruction following the current instruction to be executed next. A program (which need not contain a **halt** instruction) can terminate its computation in three ways: by executing a **halt** instruction, by executing a transfer to a nonexistent instruction (via **skip** l or **if** $p(x, y)$ **then skip** l), or by executing the last statement of the program. Two distinguished (not necessarily disjoint) sets of variables are designated input variables and output variables, respectively. We assume that all noninput variables are initialized to 0.

The main results of this paper are the following:

(1) The equivalence problem for R-programs is decidable in $2^{\lambda N^2}$ time (λ is a fixed positive constant and N is the sum of the sizes of the programs). For programs with a *fixed* number of input variables, the bound is $2^{\lambda N}$.

(2) The inequivalence problem for R-programs is in NP (= the class of languages accepted by nondeterministic polynomial-time bounded Turing machines [3]).

* Received by the editors December 19, 1979. This research was supported by the National Science Foundation under grant MCS-78-01736.

† Computer Science Department, Institute of Technology, University of Minnesota, Minneapolis, Minnesota 55455.

[1] Given two programs, are they defined at the same points and equal wherever they are defined?

[2] Given a program, does it output 1 for all inputs?

[3] $\{i_1, \cdots, i_k\}$-programs denotes the class of programs using only instructions of the form $i_1, \cdots, i_k$. x/y is integer division. (Thus, 4/3 is 1 and $-4/3$ is -1.)

(3) The equivalence problem for $\{x \leftarrow 2x,\ x \leftarrow x/2,\ x \leftarrow x+y\}$-programs with one input/output variable (i.e., the input variable is also the output variable) and one auxiliary variable is NP-hard. (The result also holds when $x \leftarrow x+y$ is replaced by $x \leftarrow x-y$.)

(4) The zero-equivalence problem (= does a program output 0 for all inputs?) for each of the following classes is NP-hard:

(i) $\{x \leftarrow 0,\ x \leftarrow 2x,\ x \leftarrow x/2,\ x \leftarrow x+y,\ x \leftarrow x-y\}$-programs with one input/output variable and one auxiliary variable.

(ii) $\{x \leftarrow 2x,\ x \leftarrow x/2,\ x \leftarrow x-y,\ x \leftarrow y\}$-programs with one input/output variable and one auxiliary variable.

(iii) $\{x \leftarrow 0,\ x \leftarrow x/2,\ x \leftarrow x-y\}$-programs with one input/output variable and two auxiliary variables.

(5) The zero-equivalence problem for each of the following classes is decidable in polynomial time.

(i) $\{x \leftarrow 0,\ x \leftarrow c,\ x \leftarrow -c,\ x \leftarrow cx,\ x \leftarrow x/c,\ x \leftarrow x+c,\ x \leftarrow x-c,\ x \leftarrow x-y\}$-programs with at most two variables. (This shows that (4) (ii)–(iii) may be the best possible results.)

(ii) $\{x \leftarrow 0,\ x \leftarrow c,\ x \leftarrow -c,\ x \leftarrow cx,\ x \leftarrow x/c,\ x \leftarrow x+c,\ x \leftarrow x-c,\ x \leftarrow x+y,\ x \leftarrow y\}$-programs (with no restriction on the number of input and auxiliary variables). This contrasts (3) and (4) (ii)–(iii).

2. An upper bound on the complexity of the equivalence problem for *R*-programs. In this section we show that the equivalence problem for R-programs is decidable in $2^{\lambda N^2}$ time (N = sum of the sizes of the programs and λ is a fixed positive constant). For programs with a *fixed* number of input variables (but no restriction on the number of output and auxiliary variables), the bound is $2^{\lambda N}$. We begin with the following lemma.

LEMMA 1. *Let P be an R-program. Assume that P has m input variables $x_1, \cdots, x_m$ and one output variable x_1. Let the other variables be $x_{m+1}, \cdots, x_n$. Let r be the number of instructions in P and K = product of all positive integer constants (i.e., c's) appearing in instructions of P. Then we can construct a collection D of systems of linear Diophantine equations*[4] *with the following properties*:

(1) *D has at most $2^n \cdot 6^r$ systems of equations.*

(2) *Let S be any system in D. Then*

(i) *S has at most $2n+5r$ equations in at most $3n+5r+1$ variables.*

(ii) *Each equation in S has at most 3 variables.*

(iii) *The maximum of the absolute values of all subdeterminants of the augmented matrix*[5] *of S is $K^2 4^{2n+5r}$.*

(iv) *S has $2m$ distinguished variables $x_1^0, \cdots, x_m^0, s_1^0, \cdots, s_m^0$, where the pair (x_i^0, s_i^0) is associated with the input variable x_i of P. s_i^0 will always have value 0 or 1. $s_i^0 = 1$ is interpreted as x_i^0 being actually negative.*

(3) *P computes a nonzero function (i.e., a function which is not zero on all inputs) if and only if one of the systems in D has a nonnegative integer solution. Moreover, if a system S in D has a nonnegative integer solution, then the values of $x_1^0, \cdots, x_m^0$ with appropriate signs attached (as given by the values of $s_1^0, \cdots, s_m^0$) when input to P will make P output a nonzero value.*

[4] A linear Diophantine equation is an equation of the form $a_1v_1 + \cdots + a_kv_k = b$, where $a_1, \cdots, a_k, b$ are (positive, negative, or zero) integer constants and $v_1, \cdots, v_k$ are integer variables.

[5] The augmented matrix of a system of equations $A\bar{y} = \bar{b}$ is A augmented by column vector $\bar{b}$.

Proof. We describe the construction of a system S in D. In the construction, we will be introducing new variables: For each $1 \leqq i \leqq n$, $x_i^0, x_i^1, x_i^2, \cdots$ and $s_i^0, s_i^1, s_i^2, \cdots$ are distinct variables associated with P's input variable x_i. For $t_i \geqq 0$, $s_i^{t_i}$ denotes the sign of the value of variable $x_i^{t_i}$, where $s_i^{t_i} = 0(1)$ denotes nonnegative (negative). Similarly, new variables $u^0, u^1, u^2, \cdots$ will be used. The construction of S involves defining for each instruction in P one or more equations describing the change that occurs in the variable that is modified by the instruction. The algorithm below forms the system S as a set of equations. The algorithm is nondeterministic with each choice giving rise to a new system.

ALGORITHM CONSTRUCT

```
$S \leftarrow \varnothing$
$q \leftarrow 0$

Add ($s_{m+1}^0 = 0, x_{m+1}^0 = 0, \cdots,$          Add to $S$ the equations which initialize
     $s_n^0 = 0, x_n^0 = 0$)                          the noninput variables to 0 and
                                                      their signs to 0 (i.e., nonnegative)

Add ($s_1^0 = 0$) **or** [Add ($s_1^0 = 1,$             Nondeterministically choose the
     $x_1^0 = u^q + 1$); $q \leftarrow q + 1$]         signs of $x_1^0, \cdots, x_m^0$. If $s_i^0$ is
  ⋮                                                   chosen to be 1, check that $x_i^0 > 0$
Add ($s_m^0 = 0$) **or** [Add ($s_m^0 = 1,$
     $x_m^0 = u^q + 1$); $q \leftarrow q + 1$]

**for** $i \leftarrow 1$ **to** $n$ **do**
  $t_i \leftarrow 0$
**end**
$p \leftarrow 1$

**while** $p \leqq r$ **do**
  **case** if $p$th instruction is
    :$x_i \leftarrow 0$: Add ($s_i^{t_i+1} = 0, x_i^{t_i+1} = 0$); $t_i \leftarrow t_i + 1$; $p \leftarrow p + 1$
    :$x_i \leftarrow c$: Add ($s_i^{t_i+1} = 0, x_i^{t_i+1} = c$); $t_i \leftarrow t_i + 1$; $p \leftarrow p + 1$
    :$x_i \leftarrow cx_i$: Add ($s_i^{t_i+1} = s_i^{t_i}, x_i^{t_i+1} = cx_i^{t_i}$); $t_i \leftarrow t_i + 1$; $p \leftarrow p + 1$
    :$x_i \leftarrow x_i + x_j$: **do** (1) **or** (2) **or** (3) **or** (4) **or** (5)
      (1) Add ($s_i^{t_i} = s_j^{t_j}, s_i^{t_i+1} = s_i^{t_i}, x_i^{t_i+1} = x_i^{t_i} + x_j^{t_j}$); $t_i \leftarrow t_i + 1$; $p \leftarrow p + 1$
      (2) Add ($s_j^{t_j} - s_i^{t_i} = 1, s_i^{t_i+1} = 0, x_i^{t_i} = x_j^{t_j} + u^q, x_i^{t_i+1} = x_i^{t_i} - x_j^{t_j}$); $t_i \leftarrow t_i + 1$;
          $q \leftarrow q + 1$; $p \leftarrow p + 1$
      (3) Add ($s_j^{t_j} - s_i^{t_i} = 1, s_i^{t_i+1} = 1, x_i^{t_i} + u^q + 1 = x_j^{t_j}, x_i^{t_i+1} = x_j^{t_j} - x_i^{t_i}$);
          $t_i \leftarrow t_i + 1$; $q \leftarrow q + 1$; $p \leftarrow p + 1$
      (4) Add ($s_i^{t_i} - s_j^{t_j} = 1, s_i^{t_i+1} = 0, x_i^{t_i} + u^q = x_j^{t_j}, x_i^{t_i+1} = x_j^{t_j} - x_i^{t_i}$); $t_i \leftarrow t_i + 1$;
          $q \leftarrow q + 1$; $p \leftarrow p + 1$
      (5) Add ($s_i^{t_i} - s_j^{t_j} = 1, s_i^{t_i+1} = 1, x_i^{t_i} = x_j^{t_j} + u^q + 1, x_i^{t_i+1} = x_i^{t_i} - x_j^{t_j}$); $t_i \leftarrow t_i + 1$;
          $q \leftarrow q + 1$; $p \leftarrow p + 1$
```

```
            :$x_i \leftarrow x_i - x_j$: Same as for $x_i \leftarrow x_i + x_j$ except that in (1), (2), (3), (4), (5) the
              first equations are replaced, respectively, by:
              $s_i^{t_i} + s_j^{t_j} = 1$
              $s_i^{t_i} + s_j^{t_j} = 0$
              $s_i^{t_i} + s_j^{t_j} = 0$
              $s_i^{t_i} = 1, s_j^{t_j} = 1$
              $s_i^{t_i} = 1, s_j^{t_j} = 1$
            :$x_i \leftarrow x_i/c$: **do** (6) **or** (7) **or** (8)
              (6) Add $(s_i^{t_i} = 0,\ s_i^{t_i+1} = 0,\ cx_i^{t_i+1} + u^q = x_i^{t_i},\ u^q + u^{q+1} = c - 1)$; $t_i \leftarrow t_i + 1$;
                  $q \leftarrow q + 2$; $p \leftarrow p + 1$
              (7) Add $(s_i^{t_i} = 1, s_i^{t_i+1} = 1, cx_i^{t_i+1} + u^q = x_i^{t_i}, u^q + u^{q+1} = c - 1, x_i^{t_i+1} = u^{q+2} + 1)$; $t_i \leftarrow t_i + 1$; $q \leftarrow q + 3$; $p \leftarrow p + 1$
              (8) Add $(s_i^{t_i} = 1, s_i^{t_i+1} = 0, x_i^{t_i} + u^q = c - 1, x_i^{t_i+1} = 0)$; $t_i \leftarrow t_i + 1$; $q \leftarrow q + 1$;
                  $p \leftarrow p + 1$
              :**skip** $l$: $p \leftarrow p + l + 1$
              :**if** $x_i > x_j$ **then skip** $l$: **do** (9) **or** (10) **or** (11) **or** (12) **or** (13) **or** (14)
              (9) Add $(s_i^{t_i} = 0, s_j^{t_j} = 0, x_i^{t_i} = x_j^{t_j} + u^q + 1)$; $q \leftarrow q + 1$; $p \leftarrow p + l + 1$
              (10) Add $(s_i^{t_i} = 0, s_j^{t_j} = 0, x_i^{t_i} + u^q = x_j^{t_j})$; $q \leftarrow q + 1$; $p \leftarrow p + 1$
              (11) Add $(s_i^{t_i} = 0, s_j^{t_j} = 1)$; $p \leftarrow p + l + 1$
              (12) Add $(s_i^{t_i} = 1, s_j^{t_j} = 0)$; $p \leftarrow p + 1$
              (13) Add $(s_i^{t_i} = 1, s_j^{t_j} = 1, x_i^{t_i} + u^q + 1 = x_j^{t_j})$; $q \leftarrow q + 1$; $p \leftarrow p + l + 1$
              (14) Add $(s_i^{t_i} = 1, s_j^{t_j} = 1, x_i^{t_i} = x_j^{t_j} + u^q)$; $q \leftarrow q + 1$; $p \leftarrow p + 1$
            :**if** $x_i \geqq x_j$ **then skip** $l$:  }
            :**if** $x_i < x_j$ **then skip** $l$:   }  Handled in a similar way as in
            :**if** $x_i \leqq x_j$ **then skip** $l$:  }  **if** $x_i > x_j$ **then skip** $l$
            :**if** $x_i = x_j$ **then skip** $l$:   }
            :**if** $x_i \neq x_j$ **then skip** $l$:  }
            :**halt**: $p \leftarrow r + 1$
          **end**
      **end**
      Add $(x_1^{t_1} = u^q + 1)$                  Insures that the final value of $x_1 \neq 0$
  **end**
```

The algorithm above is nondeterministic. Every choice gives rise to a different system S. Clearly, there are at most $2^m \cdot 6^r \leqq 2^n \cdot 6^r$ systems S. This proves property (1). That properties (2) and (3) hold is easily verified. □

Next, we state a lemma concerning "small" nonnegative integer solutions to linear Diophantine equations. The proof of the lemma can be found in [5] (see also [1]).

LEMMA 2. *Let S: $A\bar{y} = \bar{b}$ be a system of linear Diophantine equations, where A is an $m \times n$ integer matrix, $\bar{y} = (y_1, \cdots, y_n)$ is a column vector of variables, and $\bar{b} = (b_1, \cdots, b_m)$ is a column vector of integer constants. If S has a nonnegative integer solution then it has a nonnegative integer solution $\hat{y}_1, \cdots, \hat{y}_n$ such that each $\hat{y}_i \leqq 3n\Delta^2$, where Δ is the maximum of the absolute values of all subdeterminants of the augmented matrix $A\bar{b}$.*

LEMMA 3. *Let P, n, r and K be as in Lemma 1. Then P computes a nonzero function if and only if it outputs a nonzero value for some input $(x_1, \cdots, x_m)$ in which each $|x_i| \leqq 2^{\lambda N}$, where N is the size of the program and λ is a fixed positive constant.*[6]

[6] $|x_i|$ = absolute value of x_i. The size of a program is the length of its representation.

Proof. From Lemmas 1 and 2, P computes a nonzero function if and only if it outputs a nonzero value for some input $(x_1, \cdots, x_m)$ in which each $|x_i| \leqq 3(3n+5r+1)(K^2 4^{2n+5r})^2$. Now if N is the size of P then $N \geqq r$, $N \geqq n \log n$, $N \geqq \log K$. It follows that $|x_i| \leqq 2^{\lambda N}$, where λ is a fixed positive constant. □

COROLLARY 1. $2^{\lambda Nm}$ *time is sufficient to decide if an arbitrary* R*-program computes a nonzero function.*

COROLLARY 2. *Deciding if an arbitrary* R*-program computes a nonzero function can be done in nondeterministic polynomial time* (NP).

We are now ready to prove the main result of this section.

THEOREM 1. *The equivalence problem for* R*-programs with* m *input variables is decidable in* $2^{\lambda Nm}$ *time* (N = *sum of the sizes of the* 2 *programs under consideration and* λ *is a fixed positive constant*).

Proof. Let P_1 and P_2 be two R-programs. Assume that they have disjoint sets of variables. Let their input variables be $x_1, \cdots, x_m$ and $y_1, \cdots, y_m$, respectively, and their output variables be $z_1, \cdots, z_k$ and $w_1, \cdots, w_k$, respectively. Define a new program P with input variables $x_1, \cdots, x_m$ and output variable x_1 as follows:

```
$y_1 \leftarrow 0$
$y_1 \leftarrow y_1 + x_1$                         $y_1 \leftarrow x_1$
    ⋮
$y_m \leftarrow 0$
$y_m \leftarrow y_m + x_m$                         $y_m \leftarrow x_m$
________
________ }$P_1'$
________

________
________ }$P_2'$
________

**if** $z_1 \neq w_1$ **then skip** $k+1$
**if** $z_2 \neq w_2$ **then skip** $k$
    ⋮
**if** $z_k \neq w_k$ **then skip** $2$
$x_1 \leftarrow 0$
**halt**
$x_1 \leftarrow 1$
**halt**
```

For $i = 1, 2$, P_i' is P_i with each **halt** instruction replaced by **skip** l, where l is the number of instructions after the **halt** to the end of P_i. Then P computes a nonzero function if and only if P_1 is not equivalent to P_2. The result follows from Corollary 1. □

COROLLARY 3. *Equivalence of* R*-programs is decidable in* $2^{\lambda N^2}$ *time. For programs with a* **fixed** *number of input variables, the bound is* $2^{\lambda N}$.

From Corollary 2, we also have

COROLLARY 4. *The inequivalence problem for* R*-programs is in* NP.

In § 3 we will see that (in)equivalence of R-programs is NP-hard. In fact, the NP-hardness result holds for a very simple subset of R-programs.

For completeness, we mention the following result in [6] which contrasts with Corollary 3. (The input variables can assume positive, negative, or zero integer values. However, the result also applies to the case when the inputs are restricted to nonnegative integers.)

THEOREM 2.

(i) *The zero-equivalence problem for* $\{x \leftarrow 1, x \leftarrow 2x, x \leftarrow x+y, x \leftarrow x/y\}$*-programs is undecidable. The result holds even if we consider only programs that compute total functions with range* $\{0, 1\}$.

(ii) *The zero-equivalence problem for* $\{x \leftarrow 1, x \leftarrow x/2, x \leftarrow x-y, x \leftarrow x*y\}$*-programs is undecidable.*

Remark. The proof of Theorem 2 in [6] was for the one-equivalence problem (deciding if a program outputs 1 for all inputs). However, the proof can trivially be modified to apply to the zero-equivalence problem.

When there is no division, we have the following proposition.

PROPOSITION 1. *The equivalence problem for* $\{x \leftarrow 0, x \leftarrow c, x \leftarrow cx, x \leftarrow x+y, x \leftarrow x-y, x \leftarrow x*y\}$*-programs (with no restriction on the number of input, output, and auxiliary variables) is decidable.*

Proof. Let P be a program with input variables $x_1, \cdots, x_n$. Without loss of generality assume that the input variables do not appear on the left-hand sides of the instructions in P. Then the value of each output variable y at the end of the program can be represented by a polynomial $p(x_1, \cdots, x_n)$ in standard form (i.e., sum of products). Moreover, $p(x_1, \cdots, x_n)$ can be found effectively. Thus, to decide if two programs are equivalent, we find the polynomials representing their outputs. Then the programs are equivalent if and only if the polynomials representing their respective outputs are identical. (Note that this process will, in general, take exponential time since the sizes of the polynomials may grow exponentially with respect to the lengths of the programs.) □

Remark. One can easily check that all the results and proofs in this section remain valid when the inputs are restricted to nonnegative integers.

3. Two-variable $\{x \leftarrow 2x, x \leftarrow x/2, x \leftarrow x+y\}$-programs.

It is very unlikely that equivalence of R-programs can be decided in polynomial time since we can show that the problem is NP-hard (see [3], [4], [7] for definitions and motivations of the terms NP-hard, NP-complete, etc.). In fact, we can show something quite surprising: The equivalence problem for $\{x \leftarrow 2x, x \leftarrow x/2, x \leftarrow x+y\}$-programs with one input variable (which is also the output variable) and one auxiliary variable is NP-hard. This result is interesting (and counterintuitive) for the following reasons:

(1) The proofs of most NP-hard results concerning equivalence of programs (see, e.g., [2]) actually show the NP-hardness of the zero-equivalence problem. Thus, for such proofs only one program is constructed. In the case of $\{x \leftarrow 2x, x \leftarrow x/2, x \leftarrow x+y\}$-programs, zero-equivalence is clearly decidable in polynomial time. Hence, the proof that equivalence is NP-hard involves the construction of two programs.

(2) There is no instruction that can set a variable to 0 or 1. Hence, there is no way to take complements, and a reduction to the satisfiability problem for Boolean formulas cannot be done directly.

(3) Only two variables (one of which is used for input/output) are needed to show NP-hardness.

(4) The variables can assume positive, negative, or zero integer values. This makes the proof harder. Note that there are some number-theoretic problems that are NP-hard when the variables are restricted to be nonnegative but become polynomial-time solvable when there is no such restriction. For example, deciding if a system of linear Diophantine equations has a nonnegative integer solution is NP-hard [9]. However, if we are interested only in any integer solution, the problem is solvable in polynomial time [8].

THEOREM 3. *The equivalence problem for* $\{x \leftarrow 2x,\ x \leftarrow x/2,\ x \leftarrow x+y\}$-*programs with one input variable (which is also the output variable) and one auxiliary variable is* NP-*hard.*

Proof. The proof uses a well-known result that the satisfiability problem for Boolean formulas in conjunctive normal form (CNF) with at most three literals per clause is NP-hard [3]. Let $F = C_1C_2 \cdots C_m$ be a Boolean formula over variables $x_1, \cdots, x_n$, where each C_i is a disjunction (i.e., sum) of at most 3 literals. (A literal is a variable or a negation of a variable.) We shall construct two programs P_F and P'_F such that they are equivalent if and only if F is not satisfiable. P_F has input/output variable x and auxiliary variable y, and it has the following form:

Initialization
P_1
$\vdots$
P_n
Adjustment
Q_1
$\vdots$
Q_m
Finalization

The input x, which can be positive, negative, or zero, is viewed as a binary number $x = \# x_n \cdots x_1 b$, where $b, x_1, \cdots, x_n$ are binary digits and $\#$ is some (possibly negative) finite string of binary digits. (*Convention*: in this proof, $\#$ represents any (possibly negative) finite string of binary digits whose exact composition is not important.) The construction is such that P_F and P'_F agree on all inputs with $b = 0$. They disagree on some input with $b = 1$ if and only if F is satisfiable. Thus, programs P_F and P'_F will not be equivalent if and only if F is satisfiable.

Before we write the codes for the different parts of P_F, we describe a routine $Z(k, l)$ which will be used many times. The parameters k and l are positive integers.

Code for $Z(k, l)$. Let $x = \# s_{k+l} \cdots s_{k+1}s_k \cdots s_1$, where $s_1, \cdots, s_{k+l}$ are binary digits. The code $Z(k, l)$ sets $s_{k+1}, \cdots, s_{k+l}$ to 0's without changing $s_1, \cdots, s_k$. We assume that $y = 0$ or it has the same sign as x at the beginning of $Z(k, l)$.

$y \leftarrow 2^{k+l}y$	coded: $y \leftarrow 2y; \cdots; y \leftarrow 2y$ $(k+l)$ times
	$y = \#\overbrace{0 \cdots 0}^{k+l}$
$y \leftarrow y + x$	$y = \# s_{k+l} \cdots s_{k+1}s_k \cdots s_1$
$y \leftarrow y/2^k$	coded: $y \leftarrow y/2; \cdots; y \leftarrow y/2$ (k times)
$y \leftarrow 2^k y$	$y = \# s_{k+l} \cdots s_{k+1}\overbrace{0 \cdots 0}^{k}$
$x \leftarrow x + y$	$x = \# \overbrace{s_{k+l-1} \cdots s_{k+1}}^{l-1} 0 s_k \cdots s_1$

$y \leftarrow 2y$ $\qquad y = \#\overbrace{s_{k+l-1}\cdots s_{k+1}}^{l-1}\overbrace{0\cdots 0}^{k+1}$

$x \leftarrow x+y$ $\qquad x = \#\overbrace{s_{k+l-2}\cdots s_{k+1}}^{l-2}00s_k\cdots s_1$

$\vdots$

$y \leftarrow 2y$ $\qquad y = \#s_{k+1}\overbrace{0\cdots 0}^{k+l-1}$

$x \leftarrow x+y$ $\qquad y = \#\overbrace{0\cdots 0}^{l}s_k\cdots s_1$

We are now ready to write the codes for the different parts of P_F.

Code for Initialization.

$x \leftarrow 2^{3m}x$

At the end of Initialization,

$$x = \#x_n\cdots x_1bd_m^3d_m^2d_m^1\cdots d_1^3d_1^2d_1^1 = \#x_n\cdots x_1b\overbrace{0\cdots 0}^{3m}.$$

It will always be the case that at the beginning of code $P_k(1 \leqq k \leqq n)$, x has the form $x = \#x_{n-k+1}\cdots x_1bd_m^3d_m^2d_m^1\cdots d_1^3d_1^2d_1^1$ and y is either 0 or a number with the same sign as x. (Note that by convention, y is 0 at the beginning of the program since y is not an input variable.)

Code for P_k, $1 \leqq k \leqq n$.

$Z(3m+n-k+2, 3m+n-k+2)$ $\qquad x = \#\overbrace{0\cdots 0}^{3m+n-k+2}x_{n-k+1}\cdots x_1bd_m^3\cdots d_1^1$

$y \leftarrow 2^{6m+2n+2}y$ $\qquad y = \#\overbrace{0\cdots 0}^{6m+2n+2}$

$y \leftarrow y+x$

$y \leftarrow y/2^{3m+n-k+1}$ $\qquad y = \#\overbrace{0\cdots 0}^{3m+n-k+2}x_{n-k+1}$

(Let $C_{k_1}, \cdots, C_{k_r}$ be the clauses in which x_{n-k+1} appears, and assume $k_1 < \cdots < k_r$.)

$y \leftarrow 2^{3(k_1-1)}y$ $\qquad y = \#\overbrace{0\cdots 0}^{3m+n-k+2}x_{n-k+1}\overbrace{0\cdots 0}^{3(k_1-1)}$

$x \leftarrow x+y$ $\qquad d_{k_1}^2 d_{k_1}^1 \leftarrow d_{k_1}^2 d_{k_1}^1 + x_{n-k+1}$ with

$x_{n-k+1}, \cdots, x_1, b, d_m^3, d_m^2, d_m^1, \cdots,$

$d_{k_1}^3, d_{k_1-1}^3, d_{k_1-1}^2, d_{k_1-1}^1, \cdots, d_1^3, d_1^2,$

d_1^1 unchanged.

```
$y \leftarrow 2^{3(k_2-k_1)}y$
$x \leftarrow x+y$
  ⋮
$y \leftarrow 2^{3(k_r-k_{r-1})}y$
$x \leftarrow x+y$
$y \leftarrow 2^{3(m-k_r)+4}y$
$y \leftarrow y+x$                          $y = \# b d_m^3 d_m^2 d_m^1 \cdots d_1^3 d_1^2 d_1^1$
$y \leftarrow y/2^{3m}$                     $y = \# b$
$y \leftarrow 2^{3m+n-k+1}$                 $y = \# b\overbrace{0 \cdots 0}^{3m+n-k+1}$
$x \leftarrow x+y$                          $x = \# \underbrace{(x_{n-k+1}+b)}_{\substack{\parallel \\ \bar{x}_{n-k+1} \text{ if } b=1}} x_{n-k} \cdots x_1 b\, d_m^3 \cdots d_1^1$

$Z(3m+n-k+2, 3m+n-k+2)$
$y \leftarrow 2^{6m+2n+2}y$
$y \leftarrow y+x$
$y \leftarrow y/2^{3m+n-k+1}$               $y = \# \overbrace{0 \cdots 0}^{3m+n-k+2}\bar{x}_{n-k+1}$
```

(Let $C_{\bar{k}_1}, \cdots, C_{\bar{k}_s}$ be the clauses in which $\bar{x}_{n-k+1}$ appears, and assume $\bar{k}_1 < \cdots < \bar{k}_s$.)

```
$y \leftarrow 2^{3(\bar{k}_1-1)}y$
$x \leftarrow x+y$
$y \leftarrow 2^{3(\bar{k}_2-\bar{k}_1)}y$
$x \leftarrow x+y$
  ⋮
$y \leftarrow 2^{3(\bar{k}_s-\bar{k}_{s-1})}y$
$x \leftarrow x+y$
```

Clearly, at the end of P_n, $x = \# b\, d_m^3\, d_m^2\, d_m^1 \cdots d_1^3\, d_1^2\, d_1^1$ where $d_k^3 = 0$ for $1 \leqq k \leqq m$. Moreover, if $b=1$ then $d_k^2\, d_k^1 > 0$ if and only if C_k is satisfied.

Code for Adjustment.

```
$Z(3m+1, 3m)$                $x = \# \overbrace{0 \cdots 0}^{3m} b\, d_m^3 \cdots d_1^1$
$y \leftarrow 2^{6m+1}y$
$y \leftarrow y+x$
$y \leftarrow y/2^{3m}$                     $y = \# \overbrace{0 \cdots 0}^{3m} b$
```

$$\left.\begin{array}{l} x \leftarrow x+y \\ y \leftarrow 2y \\ x \leftarrow x+y \end{array}\right\} d_1^3\, d_1^2\, d_1^1 \leftarrow d_1^3\, d_1^2\, d_1^1 + bb$$

$y \leftarrow 2^2 y$

$$\left.\begin{array}{l} x \leftarrow x+y \\ y \leftarrow 2y \\ x \leftarrow x+y \end{array}\right\} d_2^3\, d_2^2\, d_2^1 \leftarrow d_2^3\, d_2^2\, d_2^1 + bb$$

$y \leftarrow 2^2 y$

$\vdots$

$$\left.\begin{array}{l} x \leftarrow x+y \\ y \leftarrow 2y \\ x \leftarrow x+y \end{array}\right\} d_m^3\, d_m^2\, d_m^1 \leftarrow d_m^3\, d_m^2\, d_m^1 + bb$$

$x \leftarrow 2^m x$

At the end of Adjustment,

$$x = \#\, d_m^3\, d_m^2\, d_m^1 \cdots d_1^3\, d_1^2\, d_1^1 \overbrace{0 \cdots 0}^{m}$$

Moreover, $d_1^3 = d_2^3 = \cdots = d_m^3 = 1$ if and only if $b = 1$ and $F = C_1 C_2 \cdots C_m$ is satisfied.

Code for Q_k, $1 \leqq k \leqq m$. When Q_k is entered, x always has the form

$$x = \#\, d_{m-k+1}^3\, d_{m-k+1}^2\, d_{m-k+1}^1 \cdots d_1^3\, d_1^2\, d_1^1\, d_m^3\, d_{m-1}^3 \cdots d_{m-k+2}^3 \overbrace{0 \cdots 0}^{m-k+1}$$

$Z(m+3(m-k+1), m+3(m-k+1))$ $\qquad$ $x = \#\overbrace{0 \cdots 0}^{m+3(m-k+1)} d_{m-k+1}^3 \cdots d_1^1 d_m^3 \cdots d_{m-k+2}^3 \overbrace{0 \cdots 0}^{m-k+1}$

$y \leftarrow 2^{2m+6(m-k+1)} y$ $\qquad$ $y = \#\overbrace{0 \cdots 0}^{m+3(m-k+1)} d_{m-k+1}^3 \cdots$

$y \leftarrow y + x$ $\qquad$ $d_1^1 d_m^3 \cdots d_{m-k+2}^3 \overbrace{0 \cdots 0}^{m-k+1}$

$y \leftarrow y/2^{m+3(m-k)+2}$ $\qquad$ $y = \#\overbrace{0 \cdots 0}^{m+3(m-k+1)} d_{m-k+1}^3$

$x \leftarrow 2^{m-k} y$

$x \leftarrow x + y$ $\qquad$ $x = \#\, d_{m-k}^3 \cdots d_1^1\, d_m^3 \cdots d_{m-k+1}^3 \overbrace{0 \cdots 0}^{m-k}$

At the end of Q_m, $x = \#\, d_m^3\, d_{m-1}^3 \cdots d_1^3$ and $d_1^3 = d_2^3 = \cdots = d_m^3 = 1$ if and only if $b = 1$ and F is satisfied.

Code for Finalization.

```
Z(m, m)                  $x = \#\overbrace{0\cdots 0}^{m} d_m^3 d_{m-1}^3 \cdots d_1^3$

$y \leftarrow 2^{2m} y$
$y \leftarrow y + x$
$y \leftarrow y/2^{m-1}$            $y = \#\overbrace{0\cdots 0}^{m} d_m^3$

$x \leftarrow x + y$
$x \leftarrow x/2^m$                $x = \# h$
```

At the end of Finalization, $x = \# h$, where $h = 0$ or 1. $h = 1$ if and only if $d_1^3 = d_2^3 = \cdots = d_m^3 = 1$, i.e., if and only if $b = 1$ and F is satisfied. It follows that P_F outputs an odd number for some input if and only if F is satisfiable.

Now let P'_F be the program obtained from P_F by adding the following instructions at the end of P_F:

```
$x \leftarrow x/2$
$x \leftarrow 2x$
```

Then P'_F is equivalent to P_F if and only if F is not satisfiable. One can easily check that the sum of the sizes (total number of instructions) of P_F and P'_F is some fixed polynomial in m and n (and, therefore, in the size of F). Hence, the construction of P_F and P'_F takes polynomial time in the size of F. Since the satisfiability problem for Boolean formulas in CNF with at most 3 literals per clause is NP-hard, the result follows. □

In Theorem 3, the instruction $x \leftarrow x + y$ can be replaced by $x \leftarrow x - y$:

COROLLARY 5. *The equivalence problem for* $\{x \leftarrow 2x, x \leftarrow x/2, x \leftarrow x - y\}$*-programs with one input variable (which is also the output variable) and one auxiliary variable is* NP*-hard.*

Proof. Replace the occurrences of instructions $x \leftarrow x + y$ and $y \leftarrow y + x$ in P_F by $x \leftarrow x - y$ and $y \leftarrow y - x$, respectively. □

When $x \leftarrow 0$, $x \leftarrow x + y$, and $x \leftarrow x - y$ are in the instruction set, we have

THEOREM 4. *The zero-equivalence problem for* $\{x \leftarrow 0, x \leftarrow 2x, x \leftarrow x/2, x \leftarrow x + y, x \leftarrow x - y\}$*-programs with one input variable and one auxiliary variable is* NP*-hard. The result holds even if the instructions* $x \leftarrow 0$ *and* $x \leftarrow x - y$ *are used exactly once in the programs.*

Proof. Let $\hat{P}_F$ be the program obtained from P_F (of Theorem 3) by adding the following instructions at the end:

```
$y \leftarrow 0$
$y \leftarrow y + x$            $y = x$
$y \leftarrow y/2$
$y \leftarrow 2y$
$x \leftarrow x - y$
```

Then $\hat{P}_F$ outputs 0 for all inputs if and only if F is not satisfiable. □

COROLLARY 6. *The zero-equivalence problem for* $\{x \leftarrow 2x,\ x \leftarrow x/2,\ x \leftarrow x - y,\ x \leftarrow y\}$*-programs with one input variable and one auxiliary variable is* NP*-hard. The result holds even if the instruction* $x \leftarrow y$ *is used exactly once in the programs.*

Proof. Replace the instructions $y \leftarrow 0$; $y \leftarrow y + x$ in the proof of Theorem 4 by $y \leftarrow x$. Then the results follows from Corollary 5. □

COROLLARY 7. *The zero-equivalence problem for* $\{x \leftarrow 0,\ x \leftarrow x/2,\ x \leftarrow x - y\}$*-programs with one input variable and two auxiliary variables is* NP*-hard.*

Proof. This follows from Theorem 4 and the observation that $x \leftarrow 2x$ and $x \leftarrow x + y$ can be coded as $z \leftarrow 0$; $z \leftarrow z - x$; $x \leftarrow x - z$ and $z \leftarrow 0$; $z \leftarrow z - y$; $x \leftarrow x - z$, respectively, z a new variable. □

Corollaries 6 and 7 may be the best possible results since we can prove the following theorem.

THEOREM 5. *The zero-equivalence problem for* $\{x \leftarrow 0,\ x \leftarrow c,\ x \leftarrow -c,\ x \leftarrow cx,\ x \leftarrow x/c,\ x \leftarrow x + c,\ x \leftarrow x - c,\ x \leftarrow x - y\}$*-programs with at most two variables (both may be input variables) is decidable in polynomial time (c is any positive integer constant).*

Proof. Let P be a program with r instructions, and let $d = \max\{c\text{'s appearing in } P\} + 1$. Let x and y be the variables of P. We consider two cases.

Case 1. P has one input variable. Let d^{3r} be the input. Let $0 \leqq k \leqq r$. Then it is easy to show (by induction on k) that the following are true at the end of k instructions:

(1) Exactly one of (a) or (b) below holds for variable z (z is either x or y):

(a) $|\text{value}(z)| \leqq d^k$ and value (z) is independent of the input.

(b) $|\text{value}(z)| \geqq d^{3r-k}$.

(2) If $|\text{value}(x)| \geqq d^{3r-k}$ and $|\text{value}(y)| \geqq d^{3r-k}$, then value (x) and value (y) have *opposite* signs.

It follows from (1) and (2) that P computes the zero-function if and only if P outputs 0 on input d^{3r}.

Case 2. P has two input variables. As in Case 1, P computes the zero-function if and only if P outputs 0 on inputs $(0, d^{3r})$ and $(d^{3r}, 0)$. □

If the instruction $x \leftarrow x - y$ is replaced by $x \leftarrow x + y$ in Theorem 5, we can prove a stronger result.

THEOREM 6. *The zero-equivalence problem for* $\{x \leftarrow 0,\ x \leftarrow c,\ x \leftarrow -c,\ x \leftarrow cx,\ x \leftarrow x/c,\ x \leftarrow x + c,\ x \leftarrow x - c,\ x \leftarrow x + y,\ x \leftarrow y\}$*-programs (with no restriction on the number of input and auxiliary variables) is decidable in polynomial time.*

Proof. Let P be a program with r instructions, and let $d = \max\{c\text{'s appearing in } P\} + 1$. Then P computes the zero-function if and only if P outputs 0 when all the input variables are set to d^r. □

For one-variable programs containing only instructions of the form $x \leftarrow 0$, $x \leftarrow 1$, $x \leftarrow 2x$ and $x \leftarrow x/2$, equivalence is decidable in polynomial time:

PROPOSITION 2. *The equivalence problem for one-variable* $\{x \leftarrow 0,\ x \leftarrow 1,\ x \leftarrow 2x,\ x \leftarrow x/2\}$*-programs is decidable in polynomial time.*

Proof. This is obvious since any program P can be reduced (in polynomial time) to one of the following forms (a, k and m are some nonnegative integer constants):

(1) $x \leftarrow a$

(2) $x \leftarrow 2^k x$

(3) $x \leftarrow x/2^k$

(4) $x \leftarrow x/2^k;\ x \leftarrow 2^m x$ □

When x is restricted to nonnegative integer inputs, we can prove a stronger result:

THEOREM 7. *The equivalence problem for one-variable* $\{x \leftarrow 0,\ x \leftarrow x+1,\ x \leftarrow 2x,\ x \leftarrow x/2\}$*-programs over nonnegative integer inputs is decidable in polynomial time.*

Proof. Any program P containing only instructions $x \leftarrow 0$, $x \leftarrow x+1$, $x \leftarrow 2x$, $x \leftarrow x/2$ can be reduced (in polynomial time) to one of the following forms (a, b, k, and m are nonnegative integers):

(1) $x \leftarrow a$
(2) $x \leftarrow 2^k x + a$
(3) $x \leftarrow x/2^k$
(4) $x \leftarrow x+a;\ x \leftarrow x/2^k$
(5) $x \leftarrow x/2^k;\ x \leftarrow 2^m x + b$
(6) $x \leftarrow x+a;\ x \leftarrow x/2^k;\ x \leftarrow 2^m x + b$

The reduction can be accomplished using the following transformations:

(a) $x \leftarrow 2^k x + a;\ x \leftarrow 2^m x + b$ reduces to $x \leftarrow 2^{k+m} x + (2^m a + b)$
(b) $x \leftarrow x/2^k;\ x \leftarrow x/2^m$ reduces to $x \leftarrow x/2^{k+m}$
(c) $x \leftarrow 2^k x + a;\ x \leftarrow x/2^m$ reduces to $x \leftarrow 2^{k-m} x + a/2^m$ if $k \geqq m$
(d) $x \leftarrow 2^k x + a;\ x \leftarrow x/2^m$ reduces to $x \leftarrow x + a/2^k;\ x \leftarrow x/2^{m-k}$ if $k < m$
(e) $x \leftarrow x/2^k;\ x \leftarrow x + a$ reduces to $x \leftarrow x + 2^k a;\ x \leftarrow x/2^k$ □

Remark. Again, all the results in this section remain valid when the inputs are restricted to nonnegative integers.

REFERENCES

[1] I. BOROSH AND L. B. TREYBIG, *Bounds on positive integral solutions of linear Diophantine equations*, Proc. Amer. Math. Soc., 55 (1976), pp. 299–304.
[2] R. L. CONSTABLE, H. B. HUNT AND S. SAHNI, *On the computational complexity of scheme equivalence*, Proc. 8th Annual Princeton Conference on Information Sciences and Systems, 1974, pp. 15–20.
[3] S. A. COOK, *The complexity of theorem-proving procedures*, Proc. 3rd Annual ACM Symposium on the Theory of Computing, 1971, pp. 151–158.
[4] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability—A Guide to the Theory of* NP-*Completeness*, W. H. Freeman, San Francisco, 1979.
[5] E. M. GURARI AND O. H. IBARRA, *An* NP-*complete number-theoretic problem*, J. Assoc. Comput. Mach. 26 (1979), pp. 567–581.
[6] O. H. IBARRA AND B. S. LEININGER, *On the simplification and equivalence problems for straight-line programs*, submitted to J. Assoc. Comput. Mach. (Available as Univ. of Minnesota, Dept. of Computer Science Technical Report 79-21, September, 1979.)
[7] R. M. KARP, *Reducibility among combinatorial problems*, Complexity of Computer Computations, R. E. Miller and J. W. Thatcher, eds., Plenum, New York, pp. 85–104.
[8] D. E. KNUTH, *The Art of Computer Programming. Vol. 2—Seminumerical Algorithms*, Addison-Wesley, Reading, MA, 1969.
[9] S. SAHNI, *Computationally related problems*, this Journal, 3 (1974), pp. 262–279.