The Complexity of the Equivalence Problem for Straight-Line Programs*

Oscar H. Ibarra and Brian S. Leininger
Department of Computer Science
University of Minnesota
Minneapolis, Minnesota 55455

Abstract

We look at several classes of straight-line programs and show that the equivalence problem is either undecidable or computationally intractable for all but the trivial classes. For example, there is no algorithm to determine if an arbitrary program (with positive, negative, or zero integer inputs) using only constructs $x \leftarrow 1$, $x \leftarrow x + y$, $x \leftarrow x/y$ (integer division) outputs 0 for all inputs. The result holds even if we consider only programs which compute total 0/1 - functions. For programs using constructs $x \leftarrow 0$, $x \leftarrow c$, $x \leftarrow cx$, $x \leftarrow x/c$, $x \leftarrow x + y$, $x \leftarrow x - y$, skip ℓ, if p(x) then skip ℓ, and halt,[1] the equivalence problem is decidable in $2^{\lambda N^2}$ time (λ is a fixed positive constant and N is the maximum of the sizes of the programs). The bound cannot be reduced to a polynomial in N unless P = NP. In fact, we prove the following rather surprising result: The equivalence problem for programs with one input/output variable and one intermediate variable using only constructs $x \leftarrow x + y$ and $x \leftarrow x/2$ is NP-hard. We also show the decidability of the equivalence problem for a certain class of programs and use this result to prove the following: Let IN be the set of natural numbers and f be any total one-to-one function from IN onto IN × IN. (f is called a pair generator. Such functions are useful in recursive function and computability theory.) Then f cannot be computed by any program using only constructs $x \leftarrow 0$, $x \leftarrow c$, $x \leftarrow x + y$, $x \leftarrow x - y$, $x \leftarrow x * y$, $x \leftarrow x/y$, skip ℓ, if p(x) then skip ℓ, and halt.

*This research was supported in part by NSF Grant MCS78-01736.

[1] c is a positive integer, ℓ is a nonnegative integer, and p(x) is a predicate of the form $x > 0$, $x \geq 0$, or $x = 0$. skip ℓ causes the $\ell + 1$st instruction following the current instruction to be executed next.

1. Introduction

It is well-known that the equivalence problem (deciding if two programs are defined at the same points and equal wherever they are defined) for programs with loops is undecidable [13,14]. In this paper, we consider simple classes of straight-line programs where the only instructions allowed are the arithmetic operations. For these classes, we can show that the equivalence problem is either undecidable or computationally intractable, even under "very restricted" use of the arithmetic operations. In fact, for most classes, the undecidability/computational intractability already apply to the zero-equivalence problem (deciding if a program outputs 0 for all inputs).

Our results should be contrasted to "positive" results (see e.g. [2,3,4,17]) 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 [2,3,4,17].

Sections 2-5 summarize our results. Two proofs are given in Section 6 to illustrate our proof techniques. Throughout, we use the following notation and convention:

(1) If $I_1, \ldots, I_r$ are instruction types (i.e. constructs), then $\{I_1, \ldots, I_r\}$ - programs denotes the class of programs using only instructions of the form $I_1, \ldots, I_r$.
(2) x/y is integer division while $x \div y$ is rational division. The result of division is undefined if y = 0. A program computes a total function if there is no input which can cause the program to divide by 0.
(3) In constructs $x \leftarrow x + y$, $x \leftarrow x - y$, $x \leftarrow x * y$, and $x \leftarrow x/y$, the variables x and y need not be distinct.
(4) We assume that all noninput variables are initialized to 0.
(5) When a result concerns zero-equivalence, we assume that the programs have exactly one output variable.

2. The Power of Division

In this section, we look at the complexity of the zero-equivalence problem for straight-line programs using only arithmetic operations. The results show that the use of division makes the zero-equivalence problem either undecidable or computationally intractable. All the results concern programs whose input variables can assume (positive, negative, or zero) integer values. However, the results also apply to the case when the inputs are restricted to nonnegative integers.

Theorem 1. The zero-equivalence problem for $\{x \leftarrow 1, x \leftarrow x + y, x \leftarrow x/y\}$ - programs is undecidable. The result holds even if we consider only programs which compute total 0/1 - functions.[2]

Corollary 1. It is undecidable to determine if an arbitrary $\{x \leftarrow 1, x \leftarrow x + y, x \leftarrow x/y\}$ - program computes a total function.

In Theorem 1, division is integer division. When we use rational division, we have

Theorem 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 which compute total functions is decidable. (The result also holds when the inputs are allowed to assume rational values.)

Theorem 2 is not true if one of the programs being considered is an arbitrary program (i.e., division by 0 can occur and the output may not be defined for all inputs). In fact, we can prove

Theorem 3. The zero-equivalence problem for $\{x \leftarrow 1, x \leftarrow x + y, x \leftarrow x \div y\}$ - programs is undecidable.

Corollary 2. It is undecidable to determine if an arbitrary $\{x \leftarrow 1, x \leftarrow x + y, x \leftarrow x \div y\}$ - program computes a total function.

We do not know whether Theorem 3 holds when the input variables are allowed to assume rational values. This problem seems very difficult. We can show that if zero-equivalence were decidable for programs with rational inputs, then we would have an algorithm to determine for an arbitrary Diophantine polynomial whether it has a solution over the rationals. (This would settle a well known open problem [8].)

When integer division is replaced by multiplication, we have the following result which contrasts Theorem 1.

Theorem 4. The equivalence problem for $\{x \leftarrow 1, x \leftarrow x + y, x \leftarrow x - y, x \leftarrow x * y\}$ - programs is decidable.

Adding $x \leftarrow x/2$ to the instruction set makes the zero-equivalence problem undecidable. In fact, we have

Theorem 5. The zero-equivalence problem for $\{x \leftarrow 1, x \leftarrow x - y, x \leftarrow x * y, x \leftarrow x/2\}$ - programs is undecidable. The result holds even if the instruction $x \leftarrow x/2$ is used exactly once in the programs.

Next, we look at programs with restricted use of multiplication and division. Specifically, let R be the instruction set $\{x \leftarrow 0, x \leftarrow c, x \leftarrow cx, x \leftarrow x/c, x \leftarrow x + y, x \leftarrow x - y,$ skip ℓ, if $p(x)$ then skip ℓ, halt$\}$ where c is any positive integer, ℓ is any nonnegative integer, and $p(x)$ is a predicate of the form $x > 0$, $x \geq 0$, or $x = 0$. skip ℓ causes the $\ell + 1$st instruction following the current instruction to be executed next. An R - 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 $p(x)$ then skip ℓ), or by executing the last statement of the program. For R-programs, equivalence is decidable in exponential time:

Theorem 6.

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

(ii) The inequivalence problem for R - programs is NP-complete. (See [1,6,11] for definitions and motivations of the terms NP-complete, NP-hard, etc.)

3. Straight-Line Programs with One Input Variable

Notation. Let S be the instruction set R (of Section 2) augmented by multiplication and division instructions, i.e., $S = R \cup \{x \leftarrow x * y, x \leftarrow x/y\}$.

From Theorems 1 and 5, we know that the zero-equivalence problem for S - programs is undecidable. However, for programs with one input variable, equivalence is decidable. In fact, we have

Theorem 7. There are fixed positive constants λ_1 and λ_2 with the following properties:

(i) Two S - programs P_1 and P_2 with one input variable (but unrestricted number of output variables and intermediate variables) are equivalent if and only if they are equivalent on all inputs x such that $|x|$ = absolute value of $x \leq 2^{2^{\lambda_1 N^2}}$.

(ii) For infinitely many N's, there are nonequivalent programs of size at most N which agree on all inputs x such that $|x| \leq 2^{2^{\lambda_2 N}}$. (This shows that the double exponential bound in (i) cannot be reduced substantially.)

Theorem 7 can be used (in a rather novel way) to prove an interesting result concerning pair generators. Let $\mathbb{N}$ be the set of natural numbers. It is well-known that there exist effectively computable total one-to-one functions from $\mathbb{N}$ onto $\mathbb{N} \times \mathbb{N}$. Such functions are called pair generators [10]. Pair generators are useful in recursive function and computability theory (see, e.g., [7,9,10,15]). Using Theorems 5 and 7 we can show that pair generators are "not easy" to compute in the following sense:

Theorem 8. No S - program can compute a pair generator.

Some pair generators can be computed by programs using instruction $x \leftarrow \sqrt{x}$ (= integral part of the square root).[3] In fact, we have

[2] A 0/1 - function is one whose range is $\{0,1\}$.

[3] We assume that $x \geq 0$.

Theorem 9. There are pair generators which can be computed by $\{x \leftarrow 1, x \leftarrow x - y, x \leftarrow x * y, x \leftarrow x/2, x \leftarrow \sqrt{x}\}$ - programs.

Combining Theorems 5 and 9, we get

Corollary 3. The zero-equivalence problem for $\{x \leftarrow 1, x \leftarrow x - y, x \leftarrow x * y, x \leftarrow x/2, x \leftarrow \sqrt{x}\}$ - programs with one input variable is undecidable.

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

4. NP-Hard Problems

It is very unlikely that the equivalence problem for R - programs or for S - programs with one input variable can be decided in polynomial time since we can show that the problem is NP-hard. In fact, we can prove a rather surprising result:

Theorem 10. The equivalence problem for $\{x \leftarrow x + y, x \leftarrow x/2\}$ - programs with one input/output variable and one intermediate variable is NP-hard.

Theorem 10 is interesting (and counter-intuitive) for the following reasons:

(1) The proofs of most NP-hard results concerning equivalence of programs (see e.g. [5]) actually show the NP-hardness of the zero-equivalence problem. Thus, for such proofs only one program is constructed. In the case of $\{x \leftarrow x + y, x \leftarrow x/2\}$ - 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 [16]. However, if we are interested only on any integer solution, the problem is solvable in polynomial time [12].

We can also prove the following theorems:

Theorem 11. The zero-equivalence problem for each of the following classes is NP-hard:

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

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

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

Theorem 12. The zero-equivalence problem for each of the following classes is decidable in polynomial time (c is any positive integer):

(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 (both may be input variables). This shows that Theorem 11 (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. This contrasts Theorem 10 and Theorem 11 (ii) - (iii).

For one-variable programs, equivalence is decidable in polynomial time:

Proposition 1. 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.

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

Theorem 13. 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.

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

5. The Power of Proper Subtraction

In this section, we consider straight-line programs which use proper subtraction[4] $x \leftarrow x \dot{-} y$ instead of 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 theorem which contrasts Theorem 4.

Theorem 14. The zero-equivalence problem for $\{x \leftarrow 1, x \leftarrow x \dot{-} y, x \leftarrow x * y\}$ - programs is undecidable.

Theorem 14 does not hold for programs with one input variable since we can prove

Theorem 15. 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 (but unrestricted number of output variables and intermediate variables) is decidable.

Although Theorem 15 is a positive result, the decision procedure is not computationally feasible, even for 2-variable programs with no multiplication:

Theorem 16. The zero-equivalence problem for $\{x \leftarrow 1, x \leftarrow x + y, x \leftarrow x \dot{-} y, x \leftarrow y \dot{-} x\}$ - programs with one input/output variable and one intermediate variable is NP-hard.

We do not know whether Theorem 15 can be extended to programs with two input variables. However, we can show that a positive answer would imply the decidability of Hilbert's tenth problem for Diophantine polynomials in two unknowns. (This would resolve an open problem in number theory [8].)

For $\{x \leftarrow 1, x \leftarrow x \dot{-} y, x \leftarrow x/y, x \leftarrow y\}$ - programs, we can prove results which contrast Theorem 1 and Corollary 1.

[4] $x \dot{-} y = \begin{cases} x - y & \text{if } x \geq y \\ 0 & \text{if } x < y \end{cases}$

Theorem 17.

(i) It is decidable to determine if an arbitrary $\{x \leftarrow 1,\ x \leftarrow x \dot{-} y,\ x \leftarrow x/y,\ x \leftarrow y\}$ - program computes a total function.

(ii) Equivalence of two $\{x \leftarrow 1,\ x \leftarrow x \dot{-} y,\ x \leftarrow x/y,\ x \leftarrow y\}$ - programs at least one of which computes a total function is decidable.

Corollary 4. The zero-equivalence problem for $\{x \leftarrow 1,\ x \leftarrow x \dot{-} y,\ x \leftarrow x/y,\ x \leftarrow y\}$ - programs is decidable.

Theorem 17 (ii) becomes false if we remove the requirement that at least one of the programs computes a total function:

Theorem 18. The equivalence problem for $\{x \leftarrow 1,\ x \leftarrow x \dot{-} y,\ x \leftarrow x/y,\ x \leftarrow y\}$ - programs is undecidable.

If $x \leftarrow 1$ is replaced by $x \leftarrow x + 1$ in Theorem 17 (ii) and Corollary 4, we get contrasting results:

Theorem 19. The zero-equivalence problem for $\{x \leftarrow x + 1,\ x \leftarrow x \dot{-} y,\ x \leftarrow x/y,\ x \leftarrow y\}$ - programs is undecidable. The result holds even if we consider only programs which compute total 0/1 - functions.

Corollary 5. It is undecidable to determine if an arbitrary $\{x \leftarrow x + 1,\ x \leftarrow x \dot{-} y,\ x \leftarrow x/y,\ x \leftarrow y\}$ - program computes a total function.

6. Proofs of Theorems 1 and 10

Theorem 1. The zero-equivalence problem for $\{x \leftarrow 1,\ x \leftarrow x + y,\ x \leftarrow x/y\}$ - programs is undecidable. The result holds even if we consider only programs which compute total 0/1 - functions.

Proof. We will use the undecidability of Hilbert's tenth problem. The problem is that of determining for any Diophantine polynomial $F(x_1,\ldots, x_r)$ whether it has a positive integer solution, i.e., whether there exist positive integers $x_1,\ldots, x_r$ such that $F(x_1,\ldots, x_r) = 0$ [8]. It is known that Hilbert's tenth problem is undecidable [8], and we will use this fact to prove our result. So let F be a Diophantine polynomial with r variables. Let

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

where $c_j > 0$ and $j_i \geq 0$ for $1 \leq j \leq n$ and $1 \leq i \leq r$. Let $K_i = \max\{j_i \mid 1 \leq j \leq n\}$ for $1 \leq i \leq r$. We will construct a program P_F with $r + 1$ input variables $x_1,\ldots, x_r$, z and 4 intermediate variables u,v,w,y such that P_F computes the zero-function if and only if F has no solution. First, we define a program segment which computes the function

$$g(z) = \begin{cases} z \text{ if } z \geq 0 \text{ or } z \leq -2 \\ 1 \text{ if } z = -1 \end{cases}$$

```
w ← 1
w ← w + z
w ← w + z          w = 2z + 1
u ← 1
u ← u/w            u = 1/(2z+1)
u ← 2u             short for u ← u + u
w ← 1
u ← u + w
z ← z/u            z = z / (2(1/(2z+1)) + 1)
```

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 (v)
```

Code for Initialization

```
x_1 ← g(x_1)
 ⋮
x_r ← g(x_r)
z ← g(z)
x_1 ← x_1 + 1      can be coded as u ← 1;
                   x_1 ← x_1 + u
 ⋮
x_r ← x_r + 1
u ← 0              can be coded as u ← 1; y ← 1;
                   y ← y + y; u ← u/y
w ← 0
```

At the end of Initialization, $x_i \geq 1$ or $x_i \leq -1$, $z \geq 0$ or $z \leq -2$ and $u = w = 0$. 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, $x_i \geq 1$, $z \geq 0$ and $z = x_1^{K_1}\ldots x_r^{K_r} - 1$.

Code for α_j, $1 \leq j \leq m$:

```
v ← 1
v ← v + z
v ← v/x_1^{K_1 - j_1}
 ⋮
v ← v/x_r^{K_r - j_r}
u ← u + v   ┐
 ⋮          ├ c_j times
u ← u + v   ┘
```

Code for β_j, $m + 1 \leq j \leq n$:

Same as for α_j with occurences of u replaced by w.

Clearly, at the end of β_n, $(2u+1)/(2w+1) > 0$ and $(2w+1)/(2u+1) > 0$ if and only if $u = w$. It is also easy to verify that $z/x_1^{K_1}\ldots x_r^{K_r} = 0$ and $(z+1)/x_1^{K_1}\ldots x_r^{K_r} > 0$ if and only if $x_1^{K_1}\ldots x_r^{K_r} \geq 1$ and $z = x_1^{K_1}\ldots x_r^{K_r} - 1$.

Code for Finalization:

```
y ← 1
u ← 2u
u ← u + y          u = 2u + 1
```

$w \leftarrow 2w$	
$w \leftarrow w + y$	$w = 2w + 1$
$v \leftarrow u$	can be coded as $v \leftarrow 1$;
	$y \leftarrow y + y$; $v \leftarrow v/y$; $v \leftarrow v + u$
$u \leftarrow u/w$	
$w \leftarrow w/v$	
$v \leftarrow z$	
$v \leftarrow v/x_1^{K_1}\ldots x_r^{K_r}$	
$y \leftarrow 1$	
$z \leftarrow z + y$	
$z \leftarrow z/x_1^{K_1}\ldots x_r^{K_r}$	

(The conditions that must be checked are: $x_1 > 0, \ldots, x_r > 0$, $v = 0$, $u > 0$, $w > 0$, $z > 0$. Note that $x_i > 0$ or $x_i < 0$.)

$x_1 \leftarrow 4x_1/(4x_1+1)$	
$x_1 \leftarrow 1/(x_1+1)$	$x_1 = \begin{cases} 1 \text{ if it was } >0 \\ 0 \text{ if it was } <0 \end{cases}$
$\vdots$	
$x_r \leftarrow 4x_r/(4x_r+1)$	
$x_r \leftarrow 1/(x_r+1)$	
$v \leftarrow 1/(3v+1)$	$v = \begin{cases} 1 \text{ if it was } 0 \\ 0 \text{ if it was not } 0 \end{cases}$
$v \leftarrow v + x_1$	
$\vdots$	
$v \leftarrow v + x_r$	$v = v + x_1 + \ldots + x_r$

(The program segment above can be coded using y as temporary variable. For example, $x_1 \leftarrow 4x_1/(4x_1+1)$ can be coded as $x_1 \leftarrow 2x_1$; $x_1 \leftarrow 2x_1$; $y \leftarrow 1$; $y \leftarrow y + x_1$; $x_1 \leftarrow x_1/y$. In what follows, the variables $x_1, \ldots, x_r$, y can be used as temporary variables.)

$u \leftarrow (2u+2)/(2u+1)$	$u = \begin{cases} 2 \text{ if it was } 0 \\ 1 \text{ if it was } >0 \\ 0 \text{ if it was } <0 \end{cases}$
$w \leftarrow (2w+2)/(2w+1)$	
$z \leftarrow (2z+2)/(2z+1)$	
$v \leftarrow v + u$	
$v \leftarrow v + w$	
$v \leftarrow v + z$	$v = v + x_1 + \ldots + x_r + u + w + z$
$v \leftarrow v/(r+4)$	$v = 0$ or 1. If $v = 1$ then its final value will be 1 if and only if $u = w = z = 1$.
$v \leftarrow 2v/(u+1)$	$v = \begin{cases} 0 \text{ if } v \text{ was } 0 \text{ or } v \text{ was } 1 \text{ and } u = 2 \\ 1 \text{ if } v \text{ was } 1 \text{ and } u = 1 \\ 2 \text{ if } v \text{ was } 1 \text{ and } u = 0 \end{cases}$
$v \leftarrow 2v/(v+1)$	
$v \leftarrow 2v/(w+1)$	
$v \leftarrow 2v/(v+1)$	
$v \leftarrow 2v/(z+1)$	

Clearly, P_F has r + 5 program variables, and P_F on input $(x_1, \ldots, x_r, z)$ outputs 0 or 1. Moreover, P_F outputs 0 for all inputs if and only if $F(x_1, \ldots, x_r)$ has no solution over the positive integers. The result now follows from the undecidability of Hilbert's tenth problem. ▨

Remark. Hilbert's tenth problem is undecidable for Diophantine polynomials with no more than r = 9 variables. Thus, Theorem 1 holds for programs with a fixed number of program variables. In fact, all the undecidable results in this paper hold for programs with a fixed number of program variables.

Theorem 10. The equivalence problem for $\{x \leftarrow x+y, x \leftarrow x/2\}$ - programs with one input/output variable and one intermediate 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 [6]. Let $F = C_1 \ldots C_m$ be a Boolean formula over variables $x_1, \ldots, x_n$, 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 intermediate 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 $x = \#x_n \ldots x_1 b$, where $b, x_1, \ldots, x_n$ 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 $x = \#s_{k+\ell} \cdots s_{k+1} s_k \cdots s_1$, where $s_1, \ldots, s_{k+\ell}$ are binary digits. The code Z(k,ℓ) sets $s_{k+1}, \ldots, s_{k+\ell}$ to 0's without changing $s_1, \ldots, s_k$. We assume that y = 0 or has the same sign as x at the beginning of Z(k,ℓ).

$y \leftarrow 2^{k+\ell}y$	$y \leftarrow 2y; \ldots; y \leftarrow 2y$ (k+ℓ) times
	Thus, $y = \#\overbrace{0\ldots0}^{k+\ell}$
$y \leftarrow y + x$	$y = \#s_{k+\ell} \cdots s_{k+1} s_k \cdots s_1$
$y \leftarrow y/2^k$	$y \leftarrow y/2; \ldots; y \leftarrow y/2$ (k times)
$y \leftarrow 2^k y$	$y = \#s_{k+\ell} \cdots s_{k+1} \overbrace{0\ldots0}^{k}$

$x \leftarrow x + y$	$x = \#s_{k+\ell-1}\cdots s_{k+1}\overbrace{0}^{\ell-1}0s_k\cdots s_1$
$y \leftarrow 2y$	$y = \#s_{k+\ell-1}\cdots s_{k+1}\overbrace{0\cdots0}^{k+1}$
$x \leftarrow x + y$	$x = \#s_{k+\ell-2}\cdots s_{k+1}\overbrace{0}^{\ell-2}0s_k\cdots s_1$
$\vdots$	
$y \leftarrow 2y$	$y = \#s_{k+1}\overbrace{0\cdots0}^{k+\ell-1}$
$x \leftarrow x + y$	$y = \#\overbrace{0\cdots0}^{\ell}s_k\cdots s_1$

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 the Initialization,

$$x = \#x_n\cdots x_1 b d_m^3 d_m^2 d_m^1 \cdots d_1^3 d_1^2 d_1^1$$

$$= \#x_n\cdots x_1 b\overbrace{0\cdots0}^{3m}$$

It will always be the case that at the beginning of code P_k $(1 \le k \le n)$, x has the form $x = \#x_{n-k+1}\cdots x_1 b d_m^3 d_m^2 d_m^1 \cdots d_1^3 d_1^2 d_1^1$ 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, $1 \le k \le n$:

$Z(3m+n-k+2,\ 3m+n-k+2)$	$x = \#\overbrace{0\cdots0}^{3m+n-k+2}x_{n-k+1}\cdots x_1 b d_m^3\cdots d_1^1$
$y \leftarrow 2^{6m+2n+2}y$	$y = \#\overbrace{0\cdots0}^{6m+2n+2}$
$y \leftarrow y + x$	
$y \leftarrow y/2^{3m+n-k+1}$	$y = \#\overbrace{0\cdots0}^{3m+n-k+2}x_{n-k+1}$

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

$y \leftarrow 2^{3(k_1-1)}y$	$y = \#\overbrace{0\cdots0}^{3m+n-k+2}x_{n-k+1}\overbrace{0\cdots0}^{3(k_1-1)}$
$x \leftarrow x + y$	$d_{k_1}^2 d_{k_1}^1 \leftarrow d_{k_1}^2 d_{k_1}^1 + x_{n-k+1}$ with $x_{n-k+1}, \ldots, x_1$, b, d_m^3, d_m^2, $d_m^1, \ldots, d_{k_1}^3$, $d_{k_1}^3-1$, $d_{k_1}^2-1$, $d_{k_1-1}^1, \ldots, d_1^3, d_1^2, d_1^1$ unchanged.
$y \leftarrow 2^{3(k_2-k_1)}y$	
$x \leftarrow x + y$	
$\vdots$	
$y \leftarrow 2^{3(k_r-k_{r-1})}y$	
$x \leftarrow x + y$	
$y \leftarrow 2^{3(m-k_r)+4}y$	
$y \leftarrow y + x$	$y = \#bd_m^3 d_m^2 d_m^1 \ldots d_1^3 d_1^2 d_1^1$
$y \leftarrow y/2^{3m}$	$y = \#b$
$y \leftarrow 2^{3m+n-k+1}$	$y = \#b\overbrace{0\cdots0}^{3m+n-k+1}$
$x \leftarrow x + y$	$x = \#(\overbrace{x_{n-k+1}}^{\bar{x}_{n-k+1}} + b)x_{n-k}\cdots x_1 b d_m^3\cdots d_1^1$ if $b = 1$
$Z(3m+n-k+2, 3m+n-k+2)$	
$y \leftarrow 2^{6m+2n+2}y$	
$y \leftarrow y + x$	
$y \leftarrow y/2^{3m+n-k+1}$	$y = \#\overbrace{0\cdots0}^{3m+n-k+2}\bar{x}_{n-k+1}$

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

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

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

Code for Adjustment:

$Z(3m+1, 3m)$	$x = \#\overbrace{0\cdots0}^{3m}bd_m^3\cdots d_1^1$
$y \leftarrow 2^{6m+1}y$	
$y \leftarrow y + x$	
$y \leftarrow y/2^{3m}$	$y = \#\overbrace{0\cdots0}^{3m}b$
$x \leftarrow x + y$ $y \leftarrow 2y$ $x \leftarrow x + y$ $y \leftarrow 2^2y$	$d_1^3 d_1^2 d_1^1 \leftarrow d_1^3 d_1^2 d_1^1 + bb$
$x \leftarrow x + y$ $y \leftarrow 2y$ $x \leftarrow x + y$ $y \leftarrow 2^2y$	$d_2^3 d_2^2 d_2^1 \leftarrow d_2^3 d_2^2 d_2^1 + bb$
$\vdots$	
$x \leftarrow x + y$ $y \leftarrow 2y$ $x \leftarrow x + y$	$d_m^3 d_m^2 d_m^1 \leftarrow d_m^3 d_m^2 d_m^1 + bb$
$x \leftarrow 2^m x$	

At the end of Adjustment,

$$x = \#d_m^3 d_m^2 d_m^1 \ldots d_1^3 d_1^2 d_1^1\overbrace{0\cdots0}^{m}$$

Moreover, $d_1^3 = d_2^3 = \ldots = d_m^3 = 1$ if and only if $b = 1$

and $F = C_1C_2\ldots C_m$ is satisfied.

Code for Q_k, $1 \le k \le m$:

When Q_k is entered, x always has the form

$$x = \#d^3_{m-k+1}d^2_{m-k+1}d^1_{m-k+1}\ldots d^3_1d^2_1d^1_1d^3_md^3_{m-1}\ldots \ldots d^3_{m-k+2}\overbrace{0\ldots0}^{m-k+1}$$

$Z(m+3(m-k+1),m+3(m-k+1))$	$x = \#\overbrace{0\ldots0}^{m+3(m-k+1)}d^3_{m-k+1}\ldots \ldots d^1_1d^3_m\ldots \ldots d^3_{m-k+2}\overbrace{0\ldots0}^{m-k+1}$
$y \leftarrow 2^{2m+6(m-k+1)}y$	
$y \leftarrow y + x$	$y = \#\overbrace{0\ldots0}^{m+3(m-k+1)}d^3_{m-k+1}\ldots \ldots d^1_1d^3_m\ldots \ldots d^3_{m-k+2}\overbrace{0\ldots0}^{m-k+1}$
$y \leftarrow y/2^{m+3(m-k)+2}$	$y = \#\overbrace{0\ldots0}^{m+3(m-k+1)}d^3_{m-k+1}$
$y \leftarrow 2^{m-k}y$	
$x \leftarrow x + y$	$x = \#d^3_{m-k}\ldots d^1_1d^3_m\ldots \ldots d^3_{m-k+1}\overbrace{0\ldots0}^{m-k}$

At the end of Q_m, $x = \#d^3_md^3_{m-1}\ldots d^3_1$ and $d^3_1 = d^3_2 =\ldots$ $\ldots= d^3_m = 1$ if and only if b = 1 and F is satisfied.

Code for Finalization:

$Z(m,m)$	$x = \#\overbrace{0\ldots0}^{m}d^3_md^3_{m-1}\ldots d^3_1$
$y \leftarrow 2^{2m}y$	
$y \leftarrow y + x$	
$y \leftarrow y/2^{m-1}$	$y = \#\overbrace{0\ldots0}^{m}d^3_m$
$x \leftarrow x + y$	
$x \leftarrow x/2^m$	$x = \#h$

At the end of Finalization, $x = \#h$, where h = 0 or 1. h = 1 if and only if $d^3_1 = d^3_2 =\ldots= d^3_m = 1$, 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 (lengths of representations) 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 interals per clause is NP-hard, the result follows. ▨

References

1. Aho, A.V., Hopcroft, J.E., and Ullman, J.D., "Design and Analysis of Computer Algorithms," Addison-Wesley, Reading, MA, 1974.
2. Aho, A.V. and Johnson, S.C., Optimal code generation for expression trees, Seventh Annual ACM Symposium on Theory of Computing, May 1975, pp. 207-217.
3. Aho, A.V. and Ullman, J.D., Optimization of straight line programs, SIAM J. Computing, 1, 1 (March 1972), pp. 1-19.
4. Bruno, J. and Sethi, R., Code generation for a one-register machine, J. ACM, 23, 3 (July 1976), pp. 502-510.
5. Constable, R.L., Hunt, H.B., and Sahni, S., On the computational complexity of scheme equivalence, Proc. Eight Annual Princeton Conference on Information Sciences and Systems (1974), pp. 15-20.
6. Cook, S.A., The complexity of theorem-proving procedures, Third Annual ACM Symposium on Theory of Computing, May 1971, pp. 151-158.
7. Davis, M., "Computability and Unsolvability," McGraw-Hill, New York, 1958.
8. Davis, M., Matijasevič, Y., and Robinson, J., Hilbert's tenth problem. Diophantine equations: Positive aspects of a negative solution, Proc. of Symposium on Pure Mathematics, 28 (1976), pp. 323-378.
9. Hennie, F., "Introduction to Computability," Addison-Wesley, Reading, MA, 1977.
10. Hopcroft, J.E. and Ullman, J.D., "Introduction to Automata Theory, Languages, and Computation," Addison-Wesley, Reading, MA, 1979.
11. Karp, R.M., Reducibility among combinatorial problems, in "Complexity of Computer Computations," R.E. Miller and J.W. Thatcher, Eds., Plenum Press, New York, 1972, pp. 85-104.
12. Knuth, D.E., "The Art of Computer Programming: Vol. 2 - Seminumerical Algorithms," Addison-Wesley, Reading, MA 1969.
13. Meyer, A. and Ritchie, D., The complexity of loop programs, Proceedings of the Twenty-Second National Conference of the ACM, Thompson Book Co., Washington, D.C. (1967), pp. 465-469.
14. Minsky, M., "Computation: Finite and Infinite Machines," Pentice-Hall Inc., Englewood Clifts, NJ, 1967.
15. Rogers, H., "Theory of Recursive Functions and Effective Computability," McGraw-Hill, New York, 1967.

16. Sahni, S., Computationally related problems, SIAM J. Computing 3 (1974), pp. 262-279.

17. Sethi, R. and Ullman, J.D., The generation of optimal code for arithmetic expressions, J. ACM, 17, 4 (October 1970), pp. 715-728.