ON THE COMPLEXITY OF SIMPLE ARITHMETIC EXPRESSIONS

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

Abstract

Let $\mathbb{E}$ be the set of all simple arithmetic expressions of the form $E(x)=xT_1...T_k$, where x is a nonnegative integer variable and each T_i is a multiplication or integer division by a positive integer constant. We investigate the complexity of the inequivalence and the bounded inequivalence problems for expressions in $\mathbb{E}$. (The bounded inequivalence problem is the problem of deciding for arbitrary expressions $E_1(x)$ and $E_2(x)$ and a positive integer ℓ whether or not $E_1(x) \neq E_2(x)$ for some nonnegative integer $x < \ell$. If $\ell = \infty$, i.e., there is no upper bound on x, the problem becomes the inequivalence problem.) We show that the inequivalence problem (or equivalently, the equivalence problem) for a large subclass of $\mathbb{E}$ is decidable in polynomial time. Whether or not the problem is decidable in polynomial time for the full class $\mathbb{E}$ remains open. We also show that the bounded inequivalence problem is NP-complete even if the divisors are restricted to be equal to 2. This last result can be used to sharpen some known NP-completeness results in the literature. Note that if division is rational division, all problems are trivially decidable in polynomial time.

1. Introduction

Let $\mathbb{E}$ be the set of all simple arithmetic expressions of the form $E(x) = xT_1...T_k$, where x is a nonnegative integer variable, $k \geq 1$, and each T_i is of the form *c or of the form /d (i.e. multiplication by a positive integer constant c or integer division by a positive integer constant d). The expression is evaluated from left to right. (For example, if $E(x) = x/3*5*3/4*7/2$, then $E(0) = E(1) = E(2) = 0$, $E(3) = E(4) = E(5) = 10$, $E(6) = 24$, etc.) It can be shown (see[6]) that the inequivalence problem for expressions in $\mathbb{E}$ (i.e. deciding for arbitrary expressions $E_1(x)$ and $E_2(x)$ whether or not $E_1(x) \neq E_2(x)$ for some nonnegative integer x) is decidable in nondeterministic polynomial time. Is there a (deterministic) polynomial time algorithm to solve the problem? (Is it NP-hard? See [4] for the definitions of NP-hard, NP-complete, etc.) This seemingly simple problem is nontrivial, and so far we have no answer. However, for a large subclass of $\mathbb{E}$, we can provide a polynomial time algorithm.

Call an expression an I-expression (I for "irreducible") if the multiplication and division operations alternate. Clearly, every expression can easily be transformed (in polynomial time) to an equivalent I-expression. Thus, finding a polynomial time algorithm for expressions in $\mathbb{E}$ is equivalent to finding a polynomial time algorithm for I-expressions. Now call an I-expression $E(x)$ a C-expression (C for "canonical") if it satisfies the following condition: If c is a multiplier in $E(x)$ and *c is not the last operation, then $\gcd(c,d) = 1$ for all divisors d in $E(x)$. (For example, x/2*2/3 is an I-expression which is not a C-expression. x/2*2, x*7/10*3/8*4 and x/3*5/4*3 are C-expressions.)

We prove in this paper that the equivalence problem for C-expressions is decidable in polynomial time. As a corollary, we show that there is a polynomial time algorithm to decide equivalence of expressions in $\mathbb{E}$ whose divisors are powers of 2. The algorithm does not generalize to the full class $\mathbb{E}$. Could it be that the inequivalence problem for the full class $\mathbb{E}$ is NP-hard? We do not know, but we believe it unlikely. However, for the bounded inequivalence problem, we can provide an answer. We show that the problem of deciding for two expressions $E_1(x)$ and $E_2(x)$ and a positive integer ℓ whether or not $E_1(x) \neq E_2(x)$ for some nonnegative integer $x < \ell$ is NP-complete. The result holds even if we restrict the divisors to be equal to 2. This result can be used to sharpen known NP-completeness results. For, example, it follows that it is NP-complete to decide inequivalence of expressions of the form $\text{rem}(x/c)T_1 \ldots T_k$, where $\text{rem}(x/c) = \text{remainder}(x/c)$ appears only at the beginning, and each T_i is of the form *c or of the form /2. This shows that inequivalence of "simple functions" as defined in [9] (see also [5]) is NP-complete, even when they are highly restricted. The NP-completeness of the bounded inequivalence problem can also be used to show that the inequivalence of L_1-programs (see [5,9]) with one input variable and three intermediate variables is NP-complete, an improvement over a result in [5].

2. Simple One-Variable Straight-Line Programs

There is a one-to-one correspondence between expressions in $\mathbb{E}$ and straight-line programs over one variable x using only constructs $x \leftarrow c * x$ and $x \leftarrow x/d$. (In the sequel, $x \leftarrow c * x$ will be abbreviated $x \leftarrow cx$.) It is trivial to translate expressions into equivalent straight-line programs and vice-versa. For example, the expression x/5*2*3/2 translates to the program $x \leftarrow x/5$; $x \leftarrow 2x$; $x \leftarrow 3x$; $x \leftarrow x/2$. For notational convenience, the results and proofs in Sections 3 and 4 are stated in terms of straight-line programs. They are easily translated to similar results concerning expressions.

Notation. In the sequel, { } encloses the permitted operations for straight-line programs. For example, $\{x \leftarrow cx,\ x \leftarrow x/2^k\}$ - programs can only use instructions of the form $x \leftarrow cx$ and $x \leftarrow x/2^k$, where c and k are any positive integer constants.

3. The Uniqueness of C-Programs

In this section, we show that two C-programs are equivalent if and only if they are identical. (This result is not true for I-programs in general.) It follows that the equivalence problem for C-programs is (trivially) decidable in polynomial time. As a corollary, we show that the equivalence problem for $\{x \leftarrow cx,\ x \leftarrow x/2^k\}$ - programs is decidable in polynomial time.

Let F be a program over $\{x \leftarrow cx,\ x \leftarrow x/d\}$, where c and d are integers ≥ 2. The number of instructions in F is denoted by length(F). For convenience, we define a program of length 0, F_0, to be a program with one "multiplication" $x \leftarrow 1x$. (This is the only program where such an instruction is allowed.) Let $\mathbb{N}$ denote the set of non-negative integers. For a given n in $\mathbb{N}$, F(n) denotes the output of F on input n. $\mathbb{N}_F$ denotes the set $\{F(n) : n \in \mathbb{N}\}$.

For given programs F and G, we say that F is equivalent to G ($F \equiv G$) if $F(n) = G(n)$ for all n in $\mathbb{N}$. We say that F is equal to G ($F = G$) if F and G are identical programs.

For a program F, F' denotes the program obtained by deleting the last instruction from F. (If length(F) ≤ 1, then $F' = F_0$.)

Definition 3.1. Let F be a program over $\{x \leftarrow cx,\ x \leftarrow x/d\}$. Let the multiplications and divisions in F be, respectively, $x \leftarrow c_1x, \ldots, x \leftarrow c_ix$ and $x \leftarrow x/d_1, \ldots, x \leftarrow x/d_j$. Then:

(a) $M_F = c_1c_2\ldots c_i$ (if $i = 0$ then $M_F = 1$).
(b) $D_F = d_1d_2\ldots d_j$ (if $j = 0$ then $D_F = 1$).
(c) $R_F = \langle M_F/D_F \rangle$. ($\langle a/b \rangle$ denotes the rational number a divided by b.)

In particular, for the program of length 0, F_0 (i.e. the program $x \leftarrow 1x$), $M_{F_0} = D_{F_0} = R_{F_0} = 1$.

The proofs of the next three lemmas are straightforward.

Lemma 3.1. Let n be a positive integer divisible by D_F. Then for each m, $F(n+m) = F(n) + F(m) = R_Fn + F(m)$.

Lemma 3.2. If $F \equiv G$, then $R_F = R_G$.

Lemma 3.3. For each program F and each positive integer n, $F(n) \leq R_Fn$. $F(n) = R_Fn$ if and only if when executing the program on input n, each time a division instruction $x \leftarrow x/d$ is encountered, the value of x is divisible by d.

An "elementary transformation" on a program F is one of the following 3 operations:

1) Replacing 2 consecutive multiplications $x \leftarrow c_1x;\ x \leftarrow c_2x$ by $x \leftarrow c_1c_2x$.
2) Replacing 2 consecutive divisions $x \leftarrow x/d_1;\ x \leftarrow x/d_2$ by $x \leftarrow x/d_1d_2$.
3) Replacing 2 consecutive instructions $x \leftarrow kcx;\ x \leftarrow x/kd$ by $x \leftarrow cx;\ x \leftarrow x/d$. (If $c = 1$ ($d = 1$) then the first (second) instruction is deleted.)

A program is irreducible (in short, an I - program) if no elementary transformation is applicable to it. It is easy to show that:

(I) Elementary transformations map programs to equivalent programs.

(II) Any program over $\{x \leftarrow cx,\ x \leftarrow x/d\}$ can be reduced by elementary transformations to an I - program in polynomial time.

(III) The multiplication and division instructions in an I - program occur in alternating order, and if $x \leftarrow cx$; $x \leftarrow x/d$ are consecutive instructions, then $\gcd(c,d) = 1$.

Lemma 3.4. Let F be an I-Program. Then $F \equiv F_0$ if and only if $F = F_0$.

Proof. Clearly, if $F = F_0$ then $F \equiv F_0$. Assume that $F \equiv F_0$. Then, by Lemma 3.2, $R_F = R_{F_0} = 1$. If the first instruction in F is $x \leftarrow x/d$ (where $d > 1$), then $F(1) = 0 \neq 1 = F_0(1)$, a contradiction. If the first instruction is $x \leftarrow cx$ (where $c > 1$), then length (F) > 1 (otherwise $R_F = c > 1$), and the second instruction is, by (III) above, $x \leftarrow x/d$, where $d > 1$ and $\gcd(c,d) = 1$. Hence, on input $n = 1$, this second instruction is encountered when the value of x is c, and $d \nmid c$ (i.e. c is not divisible by d). Therefore, by Lemma 3.3, $F(1) < R_F 1 = 1 = F_0(1)$, a contradiction. It follows that the first instruction in F must be $x \leftarrow x$, which means that $F = F_0$. ▨

An I - program F is in "canonical form" (a "C - program") if it satisfies also the following:

(IV) If $x \leftarrow cx$ is an instruction in F which is not the last instruction, then for each d such that $x \leftarrow x/d$ is an instruction in F, $\gcd(c,d) = 1$. (This is equivalent to requiring that $\gcd(c,D_F) = 1$.)

There are I - Programs which are not C - programs (example: $x \leftarrow x/2$; $x \leftarrow 2x$; $x \leftarrow x/3$). But it is not hard to see, by (III) above, that every program over $\{x \leftarrow cx,\ x \leftarrow x/2^k\}$ ($c \geq 2$, $k \geq 1$) can be reduced by elementary transformations to a C - program in polynomial time. Hence, the problem of deciding equivalence between programs over $\{x \leftarrow cx,\ x \leftarrow x/2^k\}$ can be reduced in polynomial time to the problem of deciding equivalence between C - programs.

Lemma 3.5. Let M and d be positive integers such that $d \nmid M$. Let A be the set defined by $A = \{kM/d : k \in \mathbb{N}\}$. Then $\gcd(A) = 1$.

Proof. Let $a = \gcd(A)$ and assume that $a > 1$. By definition, $M/d = ax$ for some $x \geq 0$, i.e. $M = axd + r$ for some $r < d$. Since $d \nmid M$, $r > 0$. Hence, for some k, $d \leq kr < 2d$. The integer kM/d is in A, and $kM = kaxd + kr$. Hence $kM/d = kax + 1$. Since $a > 1$, $a \nmid kM/d$, a contradiction. ▨

In the remainder of this section, unless otherwise specified, all programs are assumed to be C - programs.

Lemma 3.6. (a) If $x \leftarrow x/d$ is the last instruction in F, then $\gcd(\mathbb{N}_F) = 1$. (b) If $x \leftarrow cx$ is the last instruction in F, then $\gcd(\mathbb{N}_F) = c$.

Proof. (a) By Lemma 3.1, for each $k \in \mathbb{N}$, $F'(kD_{F'}) = kM_F$, and hence $F(kD_{F'}) = kM_F/d$. It follows that $A_F = \{kM_F/d : k \in \mathbb{N}\}$ is included in $\mathbb{N}_F$. By the definition of a C - program, $d \nmid M_F$. Hence, by Lemma 3.5, with $A = A_F$ and $M = M_F$, $\gcd(A_F) = 1$, and

hence $\gcd(\mathbb{N}_F) = 1$.
(b) If $F = F_0$ then $\mathbb{N}_F = \mathbb{N}$ and $c = 1$, so the result holds trivially. If $F \neq F_0$, then $\mathbb{N}_F = \{cn : n \in \mathbb{N}_{F'}\}$, where either $F' = F_0$ or the last instruction in F' is $x \leftarrow x/d$ for some $d > 1$. In both cases $\gcd(\mathbb{N}_{F'}) = 1$, and hence $\gcd(\mathbb{N}_F) = c$. ▨

Lemma 3.7. If $F \equiv G$ and the last instruction in F is $x \leftarrow cx$, then so is the last instruction in G.

Proof. The lemma is obvious if length(F) = 0 by Lemma 3.4. So assume that length(F) > 0. If $F \equiv G$ then clearly $\mathbb{N}_F = \mathbb{N}_G$, and hence $\gcd(\mathbb{N}_F) = \gcd(\mathbb{N}_G)$. The result now follows easily from Lemma 3.6 (b). ▨

Lemma 3.8. Let $F \equiv G$ and let the last instruction in F be $x \leftarrow x/d$. Then (a) the last instruction in G is $x \leftarrow x/e$ for some e and (b) $M_F = M_G$, $D_F = D_G$.

Proof. Part (a) follows from Lemma 3.7. Now by Lemma 3.2, $\langle M_F/D_F\rangle = \langle M_G/D_G\rangle$. Also, from part (a) and the definition of C - programs (see (IV)), $\gcd(M_F,D_F) = \gcd(M_G,D_G) = 1$. It follows that $M_F = M_G$ and $D_F = D_G$. ▨

The next lemma is obvious.

Lemma 3.9. Let k, M, a, e be integers such that $kM = ae+1$. Then $\gcd(M,a) = 1$.

Theorem 3.1. $F = G$ if and only if $F \equiv G$.

Proof. Clearly, we need only prove the "if" part. The proof is an induction on length(F) + length(G). The result is trivial if length(F) + length(G) = 0. Assume that the result is true for all F and G such that length(F) + length(G) < h where $h \geq 1$. Now consider two programs F and G such that length(F) + length(G) = h. Suppose that $F \equiv G$ but $F \neq G$. We shall derive a contradiction. Since $F \equiv G$ and $h \geq 1$, by Lemma 3.4, length(F) ≥ 1 and length(G) ≥ 1.

Case 1. The last instruction in F is $x \leftarrow cx$. By Lemma 3.7, the last instruction in G is also $x \leftarrow cx$. Since $F \neq G$, $F' \neq G'$. Hence, for some n, $F'(n) \neq G'(n)$. Then, $F(n) = cF'(n) \neq cG'(n) = G(n)$, a contradiction of $F \equiv G$.

Case 2. The last instruction in F is $x \leftarrow x/d$. Then, by Lemma 3.8(a), the last instruction in G is $x \leftarrow x/e$ for some e. Also by Lemma 3.8(b), $M_F = M_G$, $D_F = D_G$. We consider 2 subcases.

Subcase 2a. $d = e$. In this case $F' \neq G'$, and by induction hypothesis, $F'(n_0) \neq G'(n_0)$ for some n_0. Without loss of generality assume that $F'(n_0) < G'(n_0)$. By the fact that if $\gcd(a,b) = 1$, then the function $n \pmod b \to an \pmod b$ is a 1 - 1 mapping of the integers (mod b) on themselves ([3], Section 1.3), there is some k, $(k < d)$, such that $kM_F \equiv -G'(n_0) \pmod d$. Let $n_1 = n_0 + kD_{F'}$. By Lemma 3.1, $G'(n_1) = G'(n_0+kD_{F'}) = G'(n_0) + kM_F \equiv 0 \pmod d$. Let $G'(n_1) = ad$. $F'(n_1) = F'(n_0+kD_{F'}) = F'(n_0) + kM_F < G'(n_0) + kM_F = ad$. It follows that $G(n_1) = a$ and $F(n_1) < a$, a contradiction.

Subcase 2b. $d \neq e$. Without loss of generality assume that $d < e$. Let the last 2 instructions in F be $x \leftarrow cx$; $x \leftarrow x/d$. (Each of F and G is of length > 2, otherwise $D_F \neq D_G$, which, by Lemma 3.8(b), contradicts the assumption that $F \equiv G$.)

By the same consideration as in subcase 2a, there exists a k, $(k < d)$, such that

$kM_F = kM_G = ae + 1$ for some a. By Lemma 3.2, $R_F = R_G$, and hence $kD_{G'}R_F = a + 1/e$. This implies that $kD_{G'}R_{F'} = ad + d/e < ad + 1$. By Lemma 3.3, $F'(kD_{G'}) \leq kD_{G'}R_{F'}$, hence $F'(kD_{G'}) \leq ad$. By Lemma 3.9, and by the equality $kM_G = ae + 1$ above, $\gcd(M_G,a) = 1$. Since $c|M_G$, this implies that $\gcd(c,a) = 1$. By the definition of a C - program, $\gcd(c,d) = 1$. This implies that $\gcd(c,ad) = 1$, and in particular that $c \nmid ad$. But, by Lemma 3.6(b), $c|F'(kD_{G'})$. This implies that $F'(kD_{G'}) \neq ad$, and hence $F'(kD_{G'}) < ad$. It follows that $F(kD_{G'}) < a$. Since $G(kD_{G'}) = kM_G/e = (ae+1)/e = a$, we get a contradiction. ▨

From Theorem 3.1 and the fact that any $\{x \leftarrow cx, x \leftarrow x/2^k\}$ - program can be transformed into a C - program in polynomial time, we have

Theorem 3.2. The equivalence probelm for $\{x \leftarrow cx, x \leftarrow x/2^k\}$ - programs is decidable in polynomial time.

Theorem 3.1 cannot be generalized to programs which are not C - programs. In fact, we have

Proposition 3.1. There is an infinite set of distinct four-instruction I - programs which are all equivalent to the C - program $x \leftarrow x/2$; $x \leftarrow 2x$.

Proof. Let m be any odd positive integer. Then the program $x \leftarrow mx$; $x \leftarrow x/2$; $x \leftarrow 2x$; $x \leftarrow x/m$ is an I - program (but not a C - program) which is equivalent to the C - program $x \leftarrow x/2$; $x \leftarrow 2x$. ▨

Open Problem: Is the equivalence problem for I - programs decidable in polynomial time? It can be shown (see [6]) that the inequivalence problem can be decided in nondeterministic polynomial time.

4. The Bounded Inequivalence Problem for $\{x \leftarrow cx, x \leftarrow x/2\}$ - Programs

In this section, we show that the problem of deciding for two $\{x \leftarrow cx, x \leftarrow x/2\}$ - programs P and P' and a positive integer ℓ whether or not P and P' disagree on some nonnegative integer input $x < \ell$ is NP-complete. (We saw in Section 3 that when there is no upper bound on x, i.e. $\ell = \infty$, the problem is decidable in polynomial time.) This result is similar in spirit to the following theorem in [8]: The problem of deciding for positive integers m, n, and ℓ whether or not there is a positive integer $x < \ell$ such that $x^2 \equiv m \pmod{n}$ is NP-complete. (Again, if there is no upper bound on x, the problem is decidable in polynomial time.) The proof of our NP-completeness result involves an intricate coding of the satisfiability problem for Boolean formulas. That the reduction can be carried out with only one program variable using only the operations of multiplication by positive integer constants and integer division by 2 is rather surprising. We believe that this coding technique is quite interesting and can be used to prove other new NP-completeness results. (The proof of the $x^2 \equiv m \pmod{n}$ result mentioned above uses an entirely different construction.)

To simplify the discussion, we first prove the following intermediate result which is of independent interest: The satisfiability problem for Boolean formulas in conjunctive normal form (CNF) where each clause contains exactly 3 negated variables or 3 unnegated variables is NP-hard. The theorem without the "exactly three literals per clause" requirement follows directly from results of Cook [1] and Gold [2]. We state it as a lemma.

Lemma 4.1. The satisfiability problem for Boolean formulas, F', in CNF with at most three literals per clause where each clause contains either all negated variables or all unnegated variables is NP-hard.

Lemma 4.2. Let $z_1, \ldots, z_5$ be distinct variables. Let $F_3 = F_0F_1F_2$, where

F_0 = product (i.e. conjunction) of all clauses of the form

$(z_i + z_j + z_k)$, $1 \le i < j < k \le 5$,

F_1 = product of all clauses of the form

$(\bar{z}_1 + \bar{z}_j + \bar{z}_k)$, $2 \le j < k \le 5$,

F_2 = product of all clauses of the form

$(\bar{z}_2 + \bar{z}_j + \bar{z}_k)$, $1 \le j < k \le 5$, $j \ne 2$, $k \ne 2$.

Then F_3 is satisfied if and only if $z_1 = z_2 = 0$ and $z_3 = z_4 = z_5 = 1$.

Proof. Clearly, F_0 is satisfied for given values of $z_1, \ldots, z_5$ if and only if at least three variables are 1. Hence, if $z_1 = z_2 = 0$ and $z_3 = z_4 = z_5 = 1$ then F_3 is satisfied. Now suppose $F_3 = F_0F_1F_2$ is satisfied for given values of $z_1, \ldots, z_5$. Then at least three of these variables are 1. If z_1 is one of these variables and z_r and z_s are at least two others that are 1 then $(\bar{z}_1 + \bar{z}_r + \bar{z}_s)$ will make F_1 have value of 0. Hence z_1 cannot be 1. Similarly, z_2 cannot be 1. It follows that if F_3 is satisfied, then $z_1 = z_2 = 0$ and $z_3 = z_4 = z_5 = 1$. ▨

Combining Lemmas 4.1 and 4.2, we have

Theorem 4.1. The satisfiability problem for Boolean formulas in CNF with exactly three literals per clause where each clause contains either all negated variables or all unnegated variables is NP-hard.

Proof. Let $F_5 = F_3F_4$, where F_3 is the formula of Lemma 4.2 and F_4 is the formula obtained from F' of Lemma 4.1 by adding the literals z_1, z_2 to clauses with less than 3 unnegated variables and the literals $\bar{z}_3, \bar{z}_4$ to clauses with less than 3 negated variables. (We assume, of course, that $z_1, \ldots, z_5$ are variables distinct from those in F'.) It is clear that we can construct F_5 to have exactly three variables per clause with each clause containing only all negated variables or all unnegated variables. Moreover, F_5 is satisfiable if and only if F' is. ▨

The next theorem is the main result of this section. It shows that the inequivalence problem for $\{x \leftarrow cx, x \leftarrow x/2\}$ - programs over bounded inputs in NP-hard.

Theorem 4.2. It is NP-hard to determine for two $\{x \leftarrow cx, x \leftarrow x/2\}$ - programs P and P' and a positive integer ℓ whether or not P and P' disagree on some nonnegative integer input $x < \ell$.

Proof. Let $F' = C_2 \ldots C_m$ be a Boolean formula in CNF over variables $x_2, \ldots, x_n$, where each C_i is a disjunction (i.e. sum) of exactly 3 negated variables or 3 unnegated variables. By Theorem 4.1, the satisfiability problem for such formulas is NP-hard.

Let x_1 be a new variable, and let $F = C_1C_2\ldots C_m$, where $C_1 = x_1$. Then F is satisfiable if and only if F' is satisfiable. Let $\ell = 2^n$. We shall construct a program P_F such that P_F outputs an odd number for some input $x < \ell$ if and only if F is satisfiable. P'_F will 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 and P'_F will disagree on some input $x < \ell$ if and only if F is satisfiable. P_F has the following form:

$$\begin{array}{l} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_m \\ x \leftarrow x/2^k \end{array}$$

At the beginning of α_1, $x = \ldots 000x_nx_{n-1}\ldots x_2x_1$, where $x_n, x_{n-1}, \ldots, x_2, x_1$ are binary digits. We describe the tasks of $\alpha_1, \ldots, \alpha_n$, $\beta_1, \ldots, \beta_m$, omitting the details. The actual coding can be found in [7].

Each α_i is of the following form:

$$x \leftarrow ax$$
$$x \leftarrow x/2$$

After $\alpha_1 .. \alpha_n$, x looks like this:

$$\#0..0A_m0..0A_{m-1}0..0A_{m-2}\ldots A_20..0A_10..0$$

where the 0..0 strings of zeroes are sufficiently long. (# represents a string of digits whose composition is not important.) Also, A_i is a linear combination of prefixes of $x_nx_{n-1}\ldots x_2x_1$ so that the third bit from the right of A_i is a one iff clause C_i is true in the interpretation specified by $x_nx_{n-1}\ldots x_2x_1$. For example, for the clause $C_i = x_2 + x_5 + x_6$, we want A_i to be $3 + x_2 + x_5 + x_6$. If either $x_2 = 1$, $x_5 = 1$ or $x_6 = 1$, the third bit from the right of A_i is one. Now, we cannot add constant 3 so we use x_1 instead, i.e., we have $3x_1 + x_2 + x_5 + x_6$. Similarly, if the clause $C_i = \bar{x}_2 + \bar{x}_5 + \bar{x}_6$, we want A_i to be $5x_1 + x_2 + x_5 + x_6$. Finally, in order to add x_j we add $x_nx_{n-1}\ldots x_j$ and subtract $x_nx_{n-1}\ldots x_{j+1}0$. Now we cannot subtract. However, since we are only interested in the result modulo 8, we can add $7{*}x_nx_{n-1}\ldots x_{j+1}0$ instead of subtraction (since $7{*}x_nx_{n-1}\ldots x_{j+1}0 \equiv -x_nx_{n-1}\ldots x_{j+1}0 \pmod 8$). Hence a suitable nonnegative linear combination of $x_nx_{n-1}\ldots x_1$ gives us the desired result. If x looks like this:

$$\#0..0B_m 0..0B_{m-1} 0..0...0..0B_1 0..0x_n x_{n-1}..x_j$$

then a single multiplication by a suitable a, i.e., $x \leftarrow ax$, will add a multiple of the prefix $x_n x_{n-1}...x_j$ to B_i; a different multiple can be added to each B_i. Then $x \leftarrow x/2$ shifts so we have x like this:

$$\#0..0B'_m 0..0B'_{m-1} 0..0...0..0B'_1 0..0x_n x_{n-1}...x_{j+1}$$

and the operation can be repeated. (Here B'_i is B_i with some multiple of $x_n...x_j$ added.)

In a similar way, the β_i gather together the third bits of each A_i. Let b_i be the third bit from the right of A_i. Then after all β_i have executed, x looks like this:

$$\#0..0C \text{ where } C \text{ is } x_1 + b_2 + 2b_3 + 4b_4 + 8b_5 +...+ 2^{m-2}b_m.$$

Now, the 2^{m-1} bit of C will be 1 iff x_1 and all the b_i's are 1, that is, if $C_1C_2..C_m$ is satisfied. Each β_i is of the form

$$x \leftarrow x/2^s;\ x \leftarrow bx;\ x \leftarrow x/2;\ x \leftarrow cx$$

where the division shifts x right until the third bit of A_i is at the right of x. Then $x \leftarrow bx$; $x \leftarrow x/2$; $x \leftarrow cx$ adds the appropriate bit of A_i to C. This is done by adding a prefix of A_i; shifting right; and subtracting the new prefix of A_i. (The new prefix lacks the third bit.) We subtract yA_i modulo 2^m by adding $(2^r-y)A_i$ for large enough r. That is, $x \leftarrow (2^r-y)A_i x$.

The final step, $x \leftarrow x/2^k$, brings the 2^{m-1} bit of C to the right of x. This bit is 1 if $C_1C_2..C_m$ was satisfied by the assignment $x_n x_{n-1}...x_1$.

Let P'_F be P_F followed by $x \leftarrow x/2$; $x \leftarrow 2x$. Then P'_F and P_F are equivalent on x, $1 \leq x \leq 2^n$, iff F is unsatisfiable. ▨

Corollary 4.1. The problem of deciding for two $\{x \leftarrow cx,\ x \leftarrow x/2\}$ - programs P and P' and a positive integer ℓ whether or not P and P' disagree on some nonnegative integer input $x < \ell$ is NP-complete.

Proof. The problem is clearly solvable in nondeterministic polynomial time (NP). ▨

When the instruction $x \leftarrow cx$ is restricted to $c = 2$, we can prove

Proposition 4.1. The problem of deciding for two $\{x \leftarrow 2x,\ x \leftarrow x/2\}$ - programs P and P' and a positive integer ℓ whether or not P and P' disagree on some nonnegative integer input $x < \ell$ is solvable in polynomial time.

Proof. This follows from the observation that any program P can be reduced (in polynomial time) to one of the following forms (k, m are nonnegative integers):

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

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

(3) $x \leftarrow x/2^k;\ x \leftarrow 2^m x$ ▨

Adding the instruction $x \leftarrow rem(x/d)$, where $rem(x/d)$ = remainder of x divided by d makes the inequivalence problem NP-complete.

Theorem 4.3. The inequivalence problem for $\{x \leftarrow cx,\ x \leftarrow x/2,\ x \leftarrow \mathrm{rem}(x/d)\}$ - programs (over nonnegative integer inputs) is NP-complete. The result holds even if the instruction $x \leftarrow \mathrm{rem}(x/d)$ is used exactly once in the programs, and d is a power of 2.

Proof. Modify the programs P_F and P'_F of Theorem 4.2 by adding the instruction $x \leftarrow \mathrm{rem}(x/2^n)$ at the beginning. Then the modified P_F and P'_F are inequivalent if and only if F is satisfiable. Hence, the problem is NP-hard. That the problem is in NP follows from a result in [6]. However, a simple direct proof that inequivalence is in NP is the following: If F is a program, let D_F be the product of all divisors in F and all d in rem(x/d) instructions. Then two programs F and G are inequivalent if and only if they disagree on some input x, $1 \leq x \leq D_F D_G$. ▨

If $x \leftarrow \mathrm{rem}(x/d)$ is used twice, we have

Theorem 4.4. The problem of deciding if a $\{x \leftarrow cx,\ x \leftarrow x/2,\ x \leftarrow \mathrm{rem}(x/d)\}$ - program (over nonnegative integer inputs) outputs a nonzero value for some input in NP-complete. The result holds even if the instruction $x \leftarrow \mathrm{rem}(x/d)$ is used exactly twice in the programs, and in each instance, d is a power of 2.

Proof. Modify the program P_F by adding the instruction $x \leftarrow \mathrm{rem}(x/2^n)$ at the beginning and the instruction $x \leftarrow \mathrm{rem}(x/2)$ at the end. Then the new P'_F outputs a nonzero value for some input if and only if F is satisfiable. ▨

Acknowledgment

We would like to thank an anonymous referee for suggestions which improved the presentation of the proof of Theorem 4.2. This research was supported in part by NSF Grant MCS78-01736.

References

1. Cook, S., The complexity of theorem proving procedures, Conference Record of the Third ACM Symposium on Theory of Computing, (1971),151-158.

2. Gold, E., Complexity of automaton identification from given data, Information and Control, 37 (1968), 302-320.

3. Herstein, I., "Topics in Algebra," Xerox College Publishing, Lexington, MA, (1964).

4. Hopcroft, J. and Ullman, J., "Introduction to Automata Theory, Languages, and Computation," Addison-Wesley, Reading, Mass., 1979.

5. Hunt III, H., Constable, R., and Sahni, S., On the computational complexity of program scheme equivalence, SIAM Journal on Computing, 9(1980), 396-416.

6. Ibarra, O. and Leininger, B., The complexity of the equivalence problem for simple loop-free programs, to appear in SIAM Journal on Computing. (Also available as University of Minnesota Computer Science Department Tech. Report 79-23.)

7. Ibarra, O., Leininger, B., and Moran, S., On the Complexity of Simple Arithmetic Expressions, University of Minnesota Computer Science Tech. Report 80-3.

8. Manders, K. and Adleman, L., NP-complete decision problems for binary quadratics, Journal of Computer and System Sciences, 16(1978), 168-184.

9. Tsichritzis, D., The equivalence problem of simple programs, Journal of the Association for Computing Machinery, 17(1970), 729-738.