Proving Programs Terminate using Well Orderings, Ramsey Theory, and Matrices

Exposition by William Gasarch

Abstract

Many programs allow the user to input data several times during its execution. If the program runs forever the user may input data infinitely often. A program terminates if it terminates no matter what the user does.

We discuss various ways to prove that program terminates. The proofs use well orderings, Ramsey Theory, and Matrices. These techniques are used by real program checkers.

General Terms: Verification, Theory.
Keywords and Phrase: Proving programs terminate, Well Orderings, Ramsey Theory, Matrices.

1 Introduction

We describe several ways to prove that programs terminate. By this we mean terminate on *any* sequence of inputs. The methods employed are well-founded orderings, Ramsey Theory, and Matrices. This paper is self contained; it does not require knowledge of any of these topics or of Programming Languages. The methods we describe are used by real program checkers.

Our account is based on the articles of B. Cook, Podelski, and Rybalchenko [12, 13, 14, 33, 34, 35, 36] Lee, Jones, and Ben-Amram [29, 30]. Termination checkers that use the methods of B. Cook, Podelski, and Rybalchenko include ARMC [40], Loopfrog [27], and Terminator [11]. Earlier Terminator checkers that used methods from [30] are Terminlog [31] and Terminweb [9]. Termination checkers that use the methods of Lee, Jones, and Ben-Amram include ACL2 [1], AProVE [2], and Julia [3].

Convention 1.1 The statement *The Program Terminates* means that it terminates no matter what the user does. The user will be supplying inputs as the program runs; hence we are saying that the user cannot come up with some (perhaps malicious) inputs that make the program run forever.

In the summary below we refer to Programs which appear later in the paper.

1. Section 3: We prove Program 3 terminates using the well founded order $(\mathsf{N}, \leq)$. We then state Theorem 3.2 that encapsulates this kind of proof.

2. Section 4: We prove Program 4 terminates using the well founded order $(\mathsf{N} \times \mathsf{N} \times \mathsf{N} \times \mathsf{N}, <_{\text{lex}})$, where $<_{\text{lex}}$ is the lexicographic ordering, and Theorem 3.2.

3. Section 5: We prove Program 4 terminates using Ramsey's Theorem. We then state Theorems 5.5 and Theorem 5.6 which encapsulates this kind of proof.

4. Section 6: We prove Program 4 terminates using Ramsey's Theorem (via Theorem 5.5) and matrices. We then state and prove Theorems 6.4 and 6.5 that encapsulates this kind of proof.

5. Section 7: We prove Program 5 terminates using Ramsey's Theorem (via Theorem 5.6) and transition invariants. We then state Theorem 7.3 that encapsulates this kind of proof. It seems difficult to obtain a proof that Program 5 terminates without using Ramsey's Theorem.

6. Section 8: We prove Program 5 terminates using Ramsey's Theorem and Matrices. Program 5 has some properties that make this a good illustration.

7. Section 9: We prove Program 6 terminates using Ramsey's Theorem and transition invariants. Program 6 has some properties that make this a good illustration.

8. Section 10: The proofs of Theorems 5.4, 5.5, and 5.6 only used Ramsey's Theorem for Transitive colorings. We show, in three ways, that this subcase of Ramsey's theorem is strictly weaker than the full Ramsey's Theorem.

9. Section 11: We examine some cases of program termination that are decidable and some that are undecidable.

10. Section 12: We discuss open problems.

11. In the Appendix we present an interesting example by Ben-Amram.

2 Notation and Definitions

Notation 2.1

1. N is the set $\{0, 1, 2, 3, \ldots, \}$. All variables are quantified over N. For example *For all* $n \geq 1$ *means for all* $n \in \{1, 2, 3, \ldots, \}$.

2. Z is the set of integers, $\{\ldots, -2, -1, 0, 1, 2, \ldots\}$.

3. Q is the set of rationals.

4. R is the set of reals.

Notation 2.2

1. In a program the command

   $x = \textbf{Input}(\mathsf{N})$

   means that x gets an integer provided by the user.

2. More generally, if A is any set, then

   $x = \textbf{Input}(A)$

   means that x gets a value from A provided by the user.

3. If we represent the set A by listing it out we will write (for example)

   $x = \textbf{Input}(y, y+2, y+4, y+6, \ldots)$

   rather than the proper but cumbersome

   $x = \textbf{Input}(\{y, y+2, y+4, y+6, \ldots\}$

All of the programs we discuss do the following: initially the variables get values supplied by the user, then there is a **While** loop. Within the **While** loop the user can specify which one of a set of statements get executed through the use of a variable called *control*. We focus on these programs for two reasons: (1) programs of this type are a building block for more complicated programs, and (2) programs of this type can already do some things of interest. We give a very general example.

Let $n, m \in \mathsf{N}$. Let g_i as $1 \leq i \leq m$ be computable functions from Z^{n+1} to Z^n. Program 1 is a general example of the programs we will be discussing.

We define this type of program formally. We call it a *program* though it is actually a program of this restricted type. We also give intuitive comments in parenthesis.

Def 2.3

1. A *program* is a tuple (S, I, R) where the following hold.

   - S is a decidable set of states. (If $(x_1, \ldots, x_n)$ are the variables in a program and they are of types $T_1, \ldots, T_n$ then $S = T_1 \times \cdots \times T_n$.)
   - I is a decidable subset of S. (I is the set of states that the program could be in initially.)
   - $R \subseteq S \times S$ is a decidable set of ordered pairs. ($R(s,t)$ iff s satisfies the condition of the **While** loop and there is some choice of instruction that takes s to t. Note that if s does not satisfy the condition of the **While** loop then there is no t such that $R(s,t)$. This models the **While** loop termination condition.)

2. A *computation* is a (finite or infinite) sequence of states $s_1, s_2, \ldots$ such that

   - $s_1 \in I$.
   - For all i such that s_i and s_{i+1} exist, $R(s_i, s_{i+1})$.

Program 1

```
Comment: X is (x[1], ..., x[n])
Comment: The g_i are computable functions from Z^{n+1} to Z^n
X = (Input(Z), Input(Z), ..., Input(Z))
While x[1] > 0 and x[2] > 0 and ··· x[n] > 0
 control = Input(1, 2, 3, ..., m)
  if control==1
   X = g_1(X, Input(Z))
  else
  if control==2
   X = g_2(X, Input(Z))
  else
   .
   .
   .
  else
  if control==m
   X = g_m(X, Input(Z))
```

- If the sequence is finite and ends in s then there is no pair in R that begins with s. Such an s is called *terminal*.

3. A program *terminates* if every computation of it is finite.

4. A *computational segment* is a sequence of states $s_1, s_2, \ldots, s_n$ such that, for all $1 \leq i \leq n-1$, $R(s_i, s_{i+1})$. Note that we do not insist that $s_1 \in I$ nor do we insist that s_n is a terminal state.

Consider Program 2.

Program 2

```
(x, y) = (Input(Z), Input(Z))
While x > 0
        control = Input(1, 2)
        if control == 1
                (x, y) = (x + 10, y − 1)
        else
        if control == 2
                (x, y) = (y + 17, x − 2)
```

Program 2 can be defined as follows:

- $S = I = \mathsf{Z} \times \mathsf{Z}$.

Program 3

```
(x, y, z) = (Input(Z), Input(Z), Input(Z))
While x > 0 and y > 0 and z > 0
        control = Input(1, 2, 3)
        if control == 1 then
                (x, y, z) = (x + 1, y − 1, z − 1)
        else
        if control == 2 then
                (x, y, z) = (x − 1, y + 1, z − 1)
        else
        if control == 3 then
                (x, y, z) = (x − 1, y − 1, z + 1)
```

- $R = \{(x, y), (x + 10, y - 1) : x, y \geq 1\} \bigcup \{(x, y), (y + 17, x - 2) : x, y \geq 1\}$.

Def 2.4 An ordering T is *well founded* if every set has a minimal element. Note that if T is well founded then there are no infinite descending sequences of elements of T.

3 A Proof Using the Ordering $(\mathsf{N}, \leq)$

To prove that every computation of Program 3 is finite we need to find a quantity that, during every iteration of the **While** Loop, decreases. None of x, y, z qualify. However, the quantity $x + y + z$ does. We use this in our proof.

Theorem 3.1 *Every computation of Program 3 is finite.*

Proof:

Let

$$f(x, y, z) = \begin{cases} 0 \text{ if any of } x, y, z \text{ are } \leq 0; \\ x + y + z \text{ otherwise.} \end{cases} \tag{1}$$

Before every iteration of the **While** loop $f(x, y, z) > 0$. After every iteration of the **While** loop $f(x, y, z)$ has decreased. Eventually there will be an iteration such that, after it executes, $f(x, y, z) = 0$. When that happens the program terminates. ∎

The keys to the proof of Theorem 3.1 are (1) $x+y+z$ decreases with every iteration, and (2) if ever $x + y + z = 0$ the the program has terminated. There is a more general theorem lurking here, which we state below without proof. Our statement uses a different notation than the original, due to Floyd [17].

Theorem 3.2 *Let $PROG = (S, I, R)$ be a program. Assume there is a well founded order $(P, <_P)$, and a map $f : S \to P$ such that the following occurs.*

Program 4

```
(w, x, y, z) = (Input(Z), Input(Z), Input(Z), Input(Z))
While w > 0 and x > 0 and y > 0 and z > 0
        control = Input(1, 2, 3)
        if control == 1 then
                x = Input(x + 1, x + 2, ...)
                w = w − 1
        else
        if control == 2 then
                y = Input(y + 1, y + 2, ..., )
                x = x − 1
        else
        if control == 3 then
                z = Input(z + 1, z + 2, ...)
                y = y − 1
```

1. *If $R(s,t)$ then $f(t) <_P f(s)$.*
2. *If the program is in a state s such that $f(s)$ is a minimal element of P, then the program terminates.*

Then any computation of PROG is finite.

4 A Proof Using the Ordering $(\mathsf{N} \times \mathsf{N} \times \mathsf{N} \times \mathsf{N}, <_{\text{lex}})$

To prove that every computation of Program 4 is finite we need to find a quantity that, during every iteration of the **While** Loop, decreases. None of x, y, z qualify. No arithmetic combination of w, x, y, z qualifies.

Def 4.1 Let P be an ordering and $k \geq 1$. The *lexicographic ordering* on P^k is the ordering

$$(a_1, \ldots, a_k) <_{\text{lex}} (b_1, \ldots, b_k)$$

if for the least i such that $a_i \neq b_i$, $a_i < b_i$.

Example 4.2 In the ordering $(\mathsf{N}^4, <_{\text{lex}})$

$$(1, 10, 10000000000, 99999999999999) <_{\text{lex}} (1, 11, 0, 0).$$

Theorem 4.3 *Every computation of Program 4 is finite.*

Proof:

Let

$$f(w,x,y,z) = \begin{cases} (0,0,0,0) \text{ if any of } w,x,y,z \text{ are } \leq 0; \\ (w,x,y,z) \text{ otherwise.} \end{cases} \tag{2}$$

We will be concerned with the order $(\mathsf{N}^4, <_{\text{lex}})$. We use the term *decrease* with respect to $<_{\text{lex}}$.

We show that both premises of Theorem 3.2 hold.

Claim 1: In every iteration of the **While** loop $f(w,x,y,z)$ decreases.

Proof of Claim 1:

Consider an iteration of the **While** loop. There are three cases.

1. control=1: w decreases by 1, x increases by an unknown amount, y stays the same, z stays the same. Since the order is lexicographic, and w is the first coordinate, the tuple decreases no matter how much x increases.

2. control=2: w stays the same, x decreases by 1, y increases by an unknown amount, z stays the same. Since the order is lexicographic, w is the first coordinate and stays the same, and x is the second coordinate and decreases, the tuple decreases no matter how much y increases.

3. control=3: w stays the same, x stays the same, y decreases by 1, z increases by an unknown amount. This case is similar to the two other cases.

End of Proof of Claim 1

Claim 2: If $f(w,x,y,z) = (0,0,0,0)$ then the program has halted.

Proof of Claim 2:

If $f(w,x,y,z) = (0,0,0,0)$ then one of w,x,y,z is ≤ 0. Hence the **While** loop condition is not satisfied and the program halts.

End of Proof of Claim 2

By Claim 1 and 2 both premises of Theorem 3.2 are satisfied. Hence Program 4 terminates. ∎

5 A Proof Using Ramsey's Theorem

In the proof of Theorem 4.3 we showed that during every single step of Program 4 the quantity (w,x,y,z) decreased with respect to the ordering $<_{\text{lex}}$. The proof of termination was easy in that we only had to deal with one step but hard in that we had to deal with the lexicographic order on $\mathsf{N} \times \mathsf{N} \times \mathsf{N} \times \mathsf{N}$ rather than just the ordering N.

In this section we will prove that Program 4 terminates in a different way. We will not need an ordering on 4-tuples. We will only deal with w,x,y,z individually. However, we will need to prove that, for *any* computational segment, at least one of w,x,y,z decreases.

We will use the infinite Ramsey's Theorem [38] (see also [19, 20, 28]) which we state here.

Notation 5.1

1. If $n \geq 1$ then K_n is the complete graph with vertex set $V = \{1, \ldots, n\}$.
2. K_{N} is the complete graph with vertex set N.

Def 5.2 Let $c, n \geq 1$. Let G be K_n or K_{N}. Let COL be a c-coloring of the edges of G. A set of vertices V is *homogeneous with respect to COL* if all the edges between vertices in V are the same color. We will drop the *with respect to COL* if the coloring is understood.

Infinite Ramsey's Theorem:

Theorem 5.3 *Let $c \geq 1$. For every c-coloring of the the edges of K_{N} there exists an infinite homogeneous set.*

Theorem 5.4 *Every computation of Program 4 is finite.*

Proof:

We first show that for every finite computational segment one of w, x, y will decrease. There are several cases.

1. If control=1 ever occurs in the segment then w will decrease. No other case makes w increase, so we are done. In all later cases we can assume that control is never 1 in the segment.
2. If control=2 ever occurs in the segment then x decreases. Since control=1 never occurs and control=3 does not make x increase, x decreases. In all later cases we can assume that control is never 1 or 2 in the segment.
3. If control=3 is the only case that occurs in the segment then y decreases.

We show Program 4 terminates. Assume, by way of contradiction, that there is an infinite computation. Let this computation be

$$(w_1, x_1, y_1, z_1), (w_2, x_2, y_2, z_2), \ldots.$$

Since in every computational segment one of w, x, y decrease we have that, for all $i < j$, either $w_i > w_j$ or $x_i > x_j$ or $y_i > y_j$. We use this to create a coloring of the edges of K_{N}. Our colors are W, X, Y. In the coloring below each case assumes that the cases above it did not occur.

$$COL(i,j) = \begin{cases} W \text{ if } w_i > w_j; \\ X \text{ if } x_i > x_j; \\ Y \text{ if } y_i > y_j. \end{cases} \tag{3}$$

By Ramsey's Theorem there is an infinite set

$$i_1 < i_2 < i_3 < \cdots$$

such that

$$COL(i_1, i_2) = COL(i_2, i_3) = \cdots .$$

(We actually know more. We know that *all* pairs have the same color. We do not need this fact here; however, see the note after Theorem 5.6.)

Assume the color is W (the cases for X, Y are similar). Then

$$w_{i_1} > w_{i_2} > w_{i_3} > \cdots .$$

Hence eventually w must be less than 0. When this happens the program terminates. This contradicts the program not terminating. ∎

The keys to the proof of Theorem 5.4 are (1) in every computational segment one of w, x, y decreases, and (2) by Ramsey's Theorem any nonterminating computation leads to an infinite decreasing sequence in a well founded set. These ideas are from Theorem 1 of [34], though similar ideas were in [30]. The next theorem, which is a subcases of Theorem 1 of [34], captures this proof technique.

Theorem 5.5 *Let $PROG = (S, I, R)$ be a program of the form of Program 1. Note that the variables are $x[1], \ldots, x[n]$. If for all computational segment $s_1, \ldots, s_n$ there exists i such that $x[i]$ in s_1 is strictly less than $x[i]$ in s_n then any computation of $PROG$ is finite.*

To prove that a program terminates we might use some function of the variables rather than the variables themselves. The next theorem, which is a generalization of Theorem 5.5, captures this.

Theorem 5.6 *Let $PROG = (S, I, R)$ be a program. Assume that there exists well founded orderings $(P_1, <_1)$,..., $(P_m, <_m)$ and functions $f_1, \ldots, f_m$ such that $f_i : S \to P_i$. Assume the following.*

1. *For all computational segment $s_i, \ldots, s_j$ there exists a such that $f_a(s_i) >_a f_a(s_j)$.*
2. *If the program is in a state s such that, for some k, $f_k(s)$ is a minimal element of P_k, then the program terminates.*

Then any computation of $PROG$ is finite.

Proof sketch:

Assume, by way of contradiction, that $PROG$ does not terminate. Then there is an infinite computation. Let this computation be

$$(s_1, s_2, s_3, \ldots)$$

For all computational segment $s_i, \ldots, s_j$ there exists a such that $f_a(s_i) >_a f_a(s_j)$. We use this to create a coloring of the edges of K_{N}. Our colors are $1, \ldots, m$.

$$COL(i,j) = \text{ the least } a \text{ such that } f_a(s_i) >_a f_a(s_j) \ .$$

The rest of the proof is similar to the proof of Theorem 5.4. ∎

The proofs of Theorems 5.4, 5.5 and 5.6 do not need the full strength of Ramsey's Theorem. Consider Theorem 5.6. For any i, j, k if $COL(i,j) = a$ (so a is the least number such that $f_a(s_i) >_a f_a(s_j)$) and $COL(j,k) = a$ (so a is the least number such that $f_a(s_j) >_a f_a(s_k)$) then one can show $COL(i,k) = a$. Such colorings are called *transitive*. Hence we only need Ramsey's Theorem for transitive colorings. We discuss this further in Section 10.

6 A Proof Using Matrices and Ramsey's Theorem

Part of the proof of Theorem 5.4 involved showing that, for any finite computational segment of Program 4, one of w, x, y, z decreases. Can such proofs be automated? Lee, Jones, and Ben-Amram [30] developed a way to partially automate such proofs. They use matrices and Ramsey's Theorem.

We use their techniques to give a proof that Program 4 terminates. We will then discuss their general technique. Cook, Podelski, Rybalchenko have also developed a way to partially automate such proofs. We discuss this in Section 7.

Let P be a program with variables $x[1], \ldots, x[n]$ and control takes values in $\{1, \ldots, \mathrm{m}\}$. Let $1 \leq k \leq \mathrm{m}$. Let $x[1], \ldots, x[n]$ be the values of the variables before the control=k code executes and let $x[1]', \ldots, x[n]'$ be the values of the variables after. In some cases we know how $x[i]$ relates to $x[j]'$. We express this information in a matrix. The key will be that matrix multiplication (defined using $(+, \min)$ rather than $(\times, +)$) of two matrices representing pieces of code will result in a matrix that represents what happens if those pieces of code are executed one after the other.

- If it is always the case that $x[i]' \leq x[j] + L$ then the (i,j) entry of the matrix is $L \in \mathsf{Z}$.
- In all other cases the (i,j) entry is ∞. Note that this may well be most of the cases since we often do not know how $x[i]'$ and $x[j]$ relate.

Example 6.1 We describe the matrices for Program 4.

The matrix for control=1 is

$$C_1 = \begin{pmatrix} -1 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & 0 \end{pmatrix}$$

The matrix for control=2 is

$$C_2 = \begin{pmatrix} 0 & \infty & \infty & \infty \\ \infty & -1 & \infty & \infty \\ \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & 0 \end{pmatrix}$$

The matrix for control=3 is

$$C_3 = \begin{pmatrix} 0 & \infty & \infty & \infty \\ \infty & 0 & \infty & \infty \\ \infty & \infty & -1 & \infty \\ \infty & \infty & \infty & \infty \end{pmatrix}$$

We want to define matrix multiplication such that if C_1 is the matrix for control=1 and C_2 is the matrix for control=2 then C_1C_2 is the matrix for what happens if first the control=1 code is executed and then the control=2 code is executed.

Lets call the variables $x[1], \ldots, x[n]$.

1. If $C_1[i,k] = L_1$ then $x[i] \leq x[k]' + L_1$. If $C_2[k,j] = L_2$ then $x[k]' \leq x[j]'' + L_2$. Hence we know that $x[i] \leq x[j]'' + (L_1 + L_2)$. Therefore we want $C_1C_2[i,j] \leq L_1 + L_2$ Hence

$$(\forall k)[C_1C_2[i,j] \leq C_1[i,k] + C_2[k,j]].$$

   If we define $\infty + L = \infty$ and $\infty + \infty = \infty$. then this inequality is true even if if L_1 or L_2 is infinity

2. Using the above we define

$$C_1C_2[i,j] = \min_k \{C_1[i,k] + C_2[k,j]\}.$$

The following theorem, from [30], we leave for the reader.

Lemma 6.2 *Let $PROG_1$ and $PROG_2$ be programs that use the variables $x[1], \ldots, x[n]$. (We think of $PROG_1$ and $PROG_2$ as being what happens in the various control cases.) Let C_1 be the matrix that represent what is known whenever $PROG_1$ is executed. Let C_2 be the matrix that represent what is known whenever $PROG_2$ is executed. Then the matrix produce C_1C_2 as defined above represents what is known when $PROG_1$ and then $PROG_2$ are executed.*

Theorem 6.3 *Every computation of Program 4 is finite.*

Proof:

Let C_1, C_2, C_3 be the matrices that represent the cases of Control=1,2,3 in Program 4 . (These matrices are in Example 6.1). We show that the premises of Theorem 5.5 hold. To do this we prove items 0-7 below. Item 0 is easily proven directly. Items 1,2,3,4,5,6,7 are easily proven by induction on the number of matrices being multiplied.

0. $C_1C_2 = C_2C_1$, $C_1C_3 = C_3C_1$, $C_2C_3 = C_3C_2$.

1. For all $a \geq 1$
$$C_1^a = \begin{pmatrix} -a & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & 0 \end{pmatrix}$$

2. For all $b \geq 1$
$$C_2^b = \begin{pmatrix} 0 & \infty & \infty & \infty \\ \infty & -b & \infty & \infty \\ \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & 0 \end{pmatrix}$$

3. For all $c \geq 1$
$$C_3^c = \begin{pmatrix} 0 & \infty & \infty & \infty \\ \infty & 0 & \infty & \infty \\ \infty & \infty & -c & \infty \\ \infty & \infty & \infty & \infty \end{pmatrix}$$

4. For all $a, b \geq 1$
$$C_1^a C_2^b = \begin{pmatrix} -a & \infty & \infty & \infty \\ \infty & -b & \infty & \infty \\ \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & 0 \end{pmatrix}$$

5. For all $a, c \geq 1$
$$C_1^a C_3^c = \begin{pmatrix} -a & \infty & \infty & \infty \\ \infty & 0 & \infty & \infty \\ \infty & \infty & -c & \infty \\ \infty & \infty & \infty & \infty \end{pmatrix}$$

6. For all $b, c \geq 1$
$$C_2^b C_3^c = \begin{pmatrix} 0 & \infty & \infty & \infty \\ \infty & -b & \infty & \infty \\ \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty \end{pmatrix}$$

7. For $a, b, c \geq 1$

$$C_1^a C_2^b C_3^c = \begin{pmatrix} -a & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & 0 \end{pmatrix}$$

We are interested in *any* sequence of executions of control=1, control=2, and control=3. Hence we are interested in *any* product of the matrices C_1, C_2, C_3. Since the multiplication of these matrices is commutative we need only concern ourselves with $C_1^a C_2^b C_3^c$ for $a, b, c \in \mathsf{N}$. In all of the cases below $a, b, c \geq 1$.

1. C_1^a: w decreases.
2. C_2^b: x decreases.
3. C_3^c: y decreases.
4. $C_1^a C_2^b$: Both w and x decrease.
5. $C_1^a C_3^c$: Both w and y decrease.
6. $C_2^b C_3^c$: x decreases.
7. $C_1^a C_2^b C_3^c$: w decreases.

Hence, for any computational segment $s_1, \ldots, s_n$ of Program 4 either w, x, y, or z decreases. Hence by Theorem 5.5 Program 4 terminates.
∎

The keys to the proof of Theorem 6.3 are (1) represent how the old and new variables relate after one iteration with a matrix, (2) use these matrices and a type of matrix multiplication to determine that for every computational segment some variable decreases, (3) use Theorem 5.5 to conclude the program terminates.

The proof technique we used above is quite general. Lee, Jones, and Ben-Amram [30](Theorem 4) have noted the following folk theorem which captures it:

Theorem 6.4 *Let $PROG = (S, I, R)$ be a program in the form of Program 1. Let $C_1, C_2, \ldots, C_m$ be the matrices associated to control=1, ..., control=m cases. If every product of the C_i's yields a matrix with a negative integer on the diagonal then the program terminates.*

Proof: Consider computational segment $s_1, \ldots, s_n$. Let the corresponding matrices be $C_{i_1}, \ldots, C_{i_n}$. By the premise the product of these matrices has a negative integer on the diagonal. Hence some variable decreases. By Theorem 5.5 the program terminates. ∎

Theorem 6.4 leads to the following algorithm to test if a programs terminates. There is one step (alas, the important one) which we do not say how to do. If done in the obvious way it may not halt.

1. Input Program P.
2. Form matrices for all the cases of control. Let them be $C_1, \ldots, C_m$.
3. Find a finite set of types of matrices $\mathcal{M}$ such that that any product of the C_i's (allowing repeats) is in $\mathcal{M}$. (If this step is implemented by looking at all possible products until a pattern emerges then this step might not terminate.)
4. If all of the elements of $\mathcal{M}$ have some negative diagonal element then output *YES the program terminates!*
5. If not the then output *I DO NOT KNOW if the program terminates!*

If all products of matrices fit a certain pattern, as they did in the proof of Theorem 6.3, then this idea for an algorithm will terminate. Even in that case, it may output *I DON"T KNOW if the program terminates!*. However, this algorithm can be used to prove that some programs terminate, just not all. It cannot be used to prove that a program will not terminate.

Theorem 6.4 only dealt with how the variables changed. We will need a more general theorem where we look at how certain functions of the variables change. Note also the next three theorems are if-and-only-if statements.

Theorem 6.5 *Let $PROG = (S, I, R)$ be a program. The following are equivalent:*

1. *There exists functions $f_1, \ldots, f_j$ such that the following occur.*
   (a) *The associated matrices are $A_1, A_2, \ldots, A_m$ describe completely how f_i on the variables before the code is executed compares to f_j on the variables after the code is executed, in the control=1, ..., control=m cases.*
   (b) *When one of the f_i is ≤ 0 then the* **While** *loop condition does not hold so the program stops.*
   (c) *Every product of the A_i's yields a matrix with a negative integer on the diagonal.*
2. *Every computation of $PROG$ is finite.*

Theorem 6.5 also leads to an algorithm to test if programs of the type we have been considering halt. This algorithm is similar to the one that follows Theorem 6.4 and hence we omit it. Similar to that algorithm, this one does not always terminate.

The following extensions of Theorem 6.5 are known. The first one is due to Ben-Amram [4].

Theorem 6.6 *Let $PROG = (S, I, R)$ be a program. The following are equivalent:*

1. *There exists functions $f_1, \ldots, f_j$ such that the following occur.*
   (a) *Items 1.a and 1.b of Theorem 6.5 hold.*
   (b) *For all i, every column of A_i has at most one non-infinity value.*
   (c) *For every product of the A_i's there is a power of it that has a negative integer on the diagonal.*

   Then every computation of $PROG$ is finite.

The condition on the columns turns out to not be necessary. Jean-Yves Moyen [32] has shown the following.

Theorem 6.7 *Let $PROG = (S, I, R)$ be a program. The following are equivalent:*

1. *There exists $f_1, \ldots, f_j$, functions such that premises 1.a, 1.b, 1.d of Theorem 6.5 hold.*
2. *Every computation of $PROG$ is finite.*

Is there an example of a program where the matrices have a product that has no negative on the diagonal, yet by Theorem 6.7 terminates? Yes! Ben-Amram has provided us with an example and has allowed us to place it in the appendix of this paper.

7 A Proof Using Transition Invariants and Ramsey's Theorem

We proved that Program 4 terminates in three different ways. The proof in Theorem 4.3 used that $(\mathsf{N} \times \mathsf{N} \times \mathsf{N} \times \mathsf{N}, <_{\text{lex}})$ is a well founded order; however, the proof only had to deal with what happened during *one* step of Program 4. The proofs in Theorem 5.4 and 6.3 used the ordering $(\mathsf{N}, \leq)$ and Ramsey's Theorem; however, the proofs had to deal with *any* computational segment of Program 4. Which proof is easier? This is a matter of taste; however, all of the proofs are easy once you see them.

We present an example from [34] of a program (Program 5 below) where the proof of termination using Ramsey's Theorem is easy. Podelski and Rybalchenko found this proof by hand and later their termination checker found it automatically. A proof of termination using a well founded ordering seems difficult to find. Ben-Amram and Lee [5, 29] have shown that a termination proof that explicitly exhibits a well-founded order can be automatically derived when the matrices only use entries $0, -1$, and ∞. Alas, Program 5 is not of this type; however, using some manipulation Ben-Amram (unpublished) has used this result to show that Program 5 terminates. (The proof is in the Appendix.) Hence there is a proof that Program 5 terminates that uses a well-ordering; however, it was difficult to obtain.

Program 5

```
(x, y) = (Input(Z), Input(Z))
While x > 0 and y > 0
        control = Input(1, 2)
        if control == 1 then
                (x, y) = (x − 1, x)
        else
        if control == 2 then
                (x, y) = (y − 2, x + 1)
```

Theorem 7.1 *Every computation of Program 5 is finite.*

Proof:

We assume that the computational segment enters the **While** loop, else the program has already terminated.

We could try to show that, in every computational segment, either x or y decreases. This statement is true but seems hard to prove directly. Instead we show that either x or y or $x + y$ decreases. This turns out to be much easier. Intuitively we are loading our induction hypothesis. We now proceed formally.

We show that both premises of Theorem 5.6 hold with $P_1 = P_2 = P_3 = \mathsf{N}$, $f_1(x, y) = x$, $f_2(x, y) = y$, and $f_3(x, y) = x + y$. It may seem as if knowing that $x + y$ decreases you know that either x or y decreases. However, in our proof, we will *not* know which of x, y decreases. Hence we must use x, y, and $x + y$.

Claim 1: For any computational segment, one of $x, y, x + y$ decreases.

Proof of Claim 1:

We want to prove that, for all $n \geq 2$, for all computational segment of length n

$$(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n),$$

either $x_1 > x_n$ or $y_1 > y_n$ or $x_1 + y_1 > x_n + y_n$. However, we will prove something stronger. We will prove that, for all $n \geq 2$, for all computational segment of length n

$$(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n),$$

one of the following occurs.

(1) $x_1 > 0$ and $y_1 > 0$ and $x_n < x_1$ and $y_n \leq x_1$ (so x decreases),

(2) $x_1 > 0$ and $y_1 > 0$ and $x_n < y_1 - 1$ and $y_n \leq x_1 + 1$ (so $x + y$ decreases),

(3) $x_1 > 0$ and $y_1 > 0$ and $x_n < y_1 - 1$ and $y_n < y_1$ (so y decreases),

(4) $x_1 > 0$ and $y_1 > 0$ and $x_n < x_1$ and $y_n < y_1$ (so x and y both decreases, though we just need one of them).

(In the note after the proof we refer to the OR of these four statements as *the invariant.*)

We prove this by induction on n.

Base Case: $n = 2$ so we only look at one instruction.

If $(x_2, y_2) = (x_1 - 1, x_1)$ is executed then (1) holds.

If $(x_2, y_2) = (y_1 - 2, x_1 + 1)$ is executed then (2) holds.

Induction Step: We prove Claim 1 for $n+1$ assuming it for n. There are four cases, each with two subcases.

1. $x_n < x_1$ and $y_n \leq x_1$.

   (a) If $(x_{n+1}, y_{n+1}) = (x_n - 1, x_n)$ is executed then

   - $x_{n+1} = x_n - 1 < x_1 - 1 < x_1$
   - $y_{n+1} = x_n < x_1$

   Hence (1) holds.

   (b) If $(x_{n+1}, y_{n+1}) = (y_n - 2, x_n + 1)$ is executed then

   - $x_{n+1} = y_n - 2 \leq x_1 - 2 < x_1$
   - $y_{n+1} = x_n + 1 \leq x_1$

   Hence (1) holds.

2. $x_n < y_1 - 1$ and $y_n \leq x_1 + 1$

   (a) If $(x_{n+1}, y_{n+1}) = (x_n - 1, x_n)$ is executed then

   - $x_{n+1} = x_n - 1 < y_1 - 2 < y_1 - 1$
   - $y_{n+1} = x_n < y_1 - 1 < y_1$

   Hence (3) holds.

   (b) If $(x_{n+1}, y_{n+1}) = (y_n - 2, x_n + 1)$ is executed then

   - $x_{n+1} = y_n - 2 \leq x_1 - 1 < x_1$
   - $y_{n+1} = x_n < y_1$

   Hence (4) holds.

3. $x_n < y_1 - 1$ and $y_n < y_1$

   (a) If $(x_{n+1}, y_{n+1}) = (x_n - 1, x_n)$ is executed then

   - $x_{n+1} = x_n - 1 < y_1 - 2 < y_1 - 1$
   - $y_{n+1} = x_n < y_1 - 1 < y_1$.

   Hence (3) holds.

   (b) If $(x_{n+1}, y_{n+1}) = (y_n - 2, x_n + 1)$ is executed then

- $x_{n+1} = y_n - 2 < y_1 - 2 < y_1 - 1$
- $y_{n+1} = x_n < y_1 - 1 < y_1$

Hence (3) holds.

4. $x_n < x_1$ and $y_n < y_1$

   (a) If $(x_{n+1}, y_{n+1}) = (x_n - 1, x_n)$ is executed then

   - $x_{n+1} = x_n - 1 < x_1 - 1 < x_1$
   - $y_{n+1} = x_n < x_1$

   Hence (1) holds.

   (b) If $(x_{n+1}, y_{n+1}) = (y_n - 2, x_n + 1)$ is executed then

   - $x_{n+1} = y_n - 2 < y_1 - 2 < y_1 - 1$.
   - $y_{n+1} = x_n < x_1 < x_1 + 1$.

   Hence (2) holds.

We now have that, for any computational segment either x, y, or $x + y$ decreases.
End of Proof of Claim 1

The following claim is obvious.
Claim 2: If any of x, y, $x + y$ is 0 then the program terminates.
By Claims 1 and 2 the premise of Theorem 5.6 is satisfied. Hence Program 5 terminates.
∎

We can state the invariant differently. Consider the following four orderings on $\mathsf{N} \times \mathsf{N}$ and their sum.

- T_1 is the ordering $(x', y') <_1 (x, y)$ iff $x > 0$ and $y > 0$ and $x' < x$ and $y' \leq x$.
- T_2 is the ordering $(x', y') <_2 (x, y)$ iff $x > 0$ and $y > 0$ and $x' < y - 1$ and $y' \leq x + 1$.
- T_3 is the ordering $(x', y') <_3 (x, y)$ iff $x > 0$ and $y > 0$ and $x' < y - 1$ and $y' < y$.
- T_4 is the ordering $(x', y') <_4 (x, y)$ iff $x > 0$ and $y > 0$ and $x' < x$ and $y' < y$.
- $T = T_1 \cup T_2 \cup T_3 \cup T_4$. We denote this order by $<_T$.

Note that (1) each T_i is well founded, and (2) for any computational segment

$$(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$$

we have $(x_1, y_1) <_T (x_n, y_n)$
It is easy to see that these properties of T are all we needed in the proof. This is Theorem 1 of [34] which we state and prove.

Def 7.2 Let $PROG = (S, I, R)$ be a program.

1. An ordering T, which we also denote $<_T$, on $S \times S$ is *transition invariant* if for any computational segment $s_1, \ldots, s_n$ we have $s_n <_T s_1$.

2. An ordering T is *disjunctive well-founded* if there exists well founded orderings $T_1, \ldots, T_k$ such that $T = T_1 \cup \cdots \cup T_k$. Note that the T_i need not be linear orderings, they need only be well founded. This will come up in the proof of Theorem 9.1.

Theorem 7.3 *[34] Let $PROG = (S, I, R)$ be a program. Every run of $PROG$ terminates iff there exists a disjunctive well-founded transition invariant.*

Proof: We prove that if there is a disjunctive well-founded transition invariant then every run terminates. The other direction we leave to the reader.

Let $T = T_1 \cup \cdots \cup T_k$ be the disjunctive well-founded transition invariant for $PROG$. Let $<_c$ be the ordering for T_c.

Assume, by way of contradiction, that there is an infinite sequence $s_1, s_2, s_3, \ldots$, such that each $(s_i, s_{i+1}) \in R$. Define a coloring COL by, for $i < j$,

$$COL(i, j) = \text{the least } L \text{ such that } s_j <_L s_i.$$

By Ramsey's Theorem there is an infinite set

$$i_1 < i_2 < i_3 < \cdots$$

such that

$$COL(i_1, i_2) = COL(i_2, i_3) = \cdots .$$

Let that color be L. For notational readability we denote $<_L$ by $<$ and $>_L$ by $>$. We have

$$s_{i_1} > s_{i_2} > \cdots >$$

This contradicts $<$ being well founded. ∎

Note 7.4 The proof of Theorem 7.3 seems to need the full strength of Ramsey's Theorem (unlike the proof of Theorem 5.6, see the note following its proof). We give an example, due to Ben-Amram, of a program with a disjunctive well-founded transition invariant where the coloring is not transitive. Consider Program not-transitive

Program not-transitive

```
x = Input(Z)
While  x > 0
                    x = x ÷ 2
```

It clearly terminates and you can use the transition invariant $\{(x, x') : x > x'\}$ to prove it. This leads to a transitive coloring. But what if instead your transition-invariant-generator came up with the following rather odd relations instead:

1. $T_1 = \{(x, x') : x > 3x'\}$
2. $T_2 = \{(x, x') : x > x' + 1\}$

Note that $T_1 \cup T_2$ is a disjunctive well-founded transition invariant. We show that the coloring associated to $T_1 \cup T_2$ is not transitive.

- $COL(4, 2) = 2$. That is, $(4, 2) \in T_2 - T_1$.
- $COL(2, 1) = 2$. That is, $(2, 1) \in T_2 - T_1$.
- $COL((4, 1) = 1$. That is $(4, 1) \in T_1$.

Hence COL is not a transitive coloring.

Note 7.5 If in the premise of Theorem 7.3 all of the T_i's are linear (that is, every pair of elements is comparable) then the transitive Ramsey Theorem suffices for the proof.

Finding an appropriate T is the key to the proofs of termination for the termination checkers Loopfrog [27], and Terminator [11].

8 Another Proof Using Matrices and Ramsey's Theorem

We prove Program 5 terminates using matrices. The case control=1 is represented by the matrix

$$C_1 = \begin{pmatrix} -1 & 0 \\ \infty & \infty \end{pmatrix}.$$

The case control=2 is represented by the matrix

$$C_2 = \begin{pmatrix} \infty & -2 \\ 1 & \infty \end{pmatrix}.$$

This will not work! Note that C_2 is has no negative numbers on its diagonal. Hence we cannot use these matrices in our proof! What will we do!? Instead of using x, y we will use x, y, and $x + y$. We comment on whether or not you can somehow use C_1 and C_2 after the proof.

Theorem 8.1 *Every computation of Program 5 is finite.*

Proof: We will use Theorem 6.5 with functions x, y, and $x + y$. Note that $x + y$ is not one of the original variables which is why we need Theorem 6.5 rather than Theorem 6.4.

The control=1 case of Program 5 corresponds to

$$D_1 = \begin{pmatrix} -1 & 0 & 1 \\ \infty & \infty & \infty \\ \infty & \infty & \infty \end{pmatrix}$$

The control=2 case of Program 5 corresponds to

$$D_2 = \begin{pmatrix} \infty & 1 & \infty \\ -2 & \infty & \infty \\ \infty & \infty & -1 \end{pmatrix}$$

We show that the premises of Theorem 6.5 hold. The following are true and easily proven by induction on the number of matrices being multiplied.

1. For all $a \geq 1$

$$D_1^a = \begin{pmatrix} -a & -a+1 & -a+2 \\ \infty & \infty & \infty \\ \infty & \infty & \infty \end{pmatrix}$$

2. For all $b \geq 1$, b odd, $b = 2d - 1$,

$$D_2^b = \begin{pmatrix} -d & \infty & \infty \\ \infty & -d & \infty \\ \infty & \infty & -2d \end{pmatrix}$$

3. For all $b \geq 2$, b even, $b = 2e$,

$$D_2^b = \begin{pmatrix} \infty & -e+1 & \infty \\ -e-2 & \infty & \infty \\ \infty & \infty & -2e-1 \end{pmatrix}$$

4. For all $a, b \geq 1$, b odd, $b = 2d - 1$.

$$D_1^a D_2^b = \begin{pmatrix} -a-d & -a-d+1 & -a-2d+2 \\ \infty & \infty & \infty \\ \infty & \infty & \infty \end{pmatrix}$$

5. For all $a, b \geq 1$, b even, $b = 2e$.

$$D_1^a D_2^b = \begin{pmatrix} -a-e-1 & -a-e+1 & -a-2e+1 \\ \infty & \infty & \infty \\ \infty & \infty & \infty \end{pmatrix}$$

6. For all $a, b \geq 1$, a is odd,

$$D_2^a D_1^b = \begin{pmatrix} \infty & \infty & \infty \\ -(\lfloor a/2 \rfloor + b + 2 & -(\lfloor a/2 \rfloor + b + 1 & -(\lfloor a/2 \rfloor + b \\ \infty & \infty & \infty \end{pmatrix}$$

7. If $a, b \geq 1$, a is even,

$$D_2^a D_1^b = \begin{pmatrix} -(a/2) + b & -(a/2) + b - 1 & -\lfloor a/2 \rfloor + b - 2 \\ \infty & \infty & \infty \\ \infty & \infty & \infty \end{pmatrix}$$

We use this information to formulate a lemma.

Convention: If we put < 0 (≤ 0) in an entry of a matrix it means that the entry is some integer less than 0 (less than or equal to 0). We might not know what it is.

Claim: For all $n \geq 2$, any product of n matrices all of which are D_1's and D_2's must be of one of the following type:

1.
$$\begin{pmatrix} < 0 & \leq 0 & \leq 0 \\ \infty & \infty & \infty \\ \infty & \infty & \infty \end{pmatrix}$$

2.
$$\begin{pmatrix} \infty & \infty & \infty \\ < 0 & < 0 & < 0 \\ \infty & \infty & \infty \end{pmatrix}$$

3.
$$\begin{pmatrix} < 0 & \infty & \infty \\ \infty & < 0 & \infty \\ \infty & \infty & < 0 \end{pmatrix}$$

4.
$$\begin{pmatrix} \infty & < 0 & \infty \\ < 0 & \infty & \infty \\ \infty & \infty & < 0 \end{pmatrix}$$

End of Claim

This can be proved easily by induction on n. ∎

One can show that every computation of Program 5 terminates using the original matrices 2×2 matrices C_1, C_2. This requires a more advanced theorem (Theorem 6.7 above). Ben-Amram has done this and has allowed us to place his proof in the appendix of this paper.

9 Another Proof using Transition Invariants and Ramsey's Theorem

Showing Program 6 terminates seems easy: eventually y is negative and after that point x will steadily decrease until $x < 0$. But this proof might be hard for a termination checker to find since x might increases for a very long time. Instead we need to find the right disjunctive well-founded transition invariant.

Program 6

$(x, y) = (\textbf{Input}(\mathsf{Z}), \textbf{Input}(\mathsf{Z}))$
While $x > 0$
$(x, y) = (x + y, y - 1)$

Theorem 9.1 *Every run of Program 6 terminates.*

Proof: We define orderings T_1 and T_2 which we also denote $<_1$ and $<_2$.

- $(x', y') <_1 (x, y))$ iff $0 < x$ and $x' < x$.
- $(x', y') <_2 (x, y))$ iff $0 \leq y$ and $y' < y'$.

Let

$$T = T_1 \cup T_2.$$

Clearly T_1 and T_2 are well-founded (though see note after the proof). Hence T is disjunctive well-founded. We show that T is a transition invariant.

We want to prove that, for all $n \geq 2$, for all computational segment of length n

$$(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$$

either T_1 or T_2 holds.

We prove this by induction on n.

We will assume that the computational segment enters the **While** loop, else the program has already terminated. In particular, in the base case, $x > 0$.

Base Case: $n = 2$ so we only look at one instruction. There are two cases.

If $y_1 \geq 0$ then $y_2 = y_1 - 1 < y_1$. Hence $(x_2, y_2) <_2 (x_1, y_1)$ whatever x_1, x_2 are.

If $y_1 < 0$ then $x_2 = x_1 + y_1 < x_1$. Hence $(x_2, y_2) <_1 (x_1, y_1)$ whatever x_1, x_2 are.

Induction Step: There are four cases based on (1) $y \leq 0$ or $y > 0$, and (2) $<_1$ or $<_2$ holds between (x_1, y_1) and (x_n, y_n). We omit details. ∎

T_1 and T_2 are *partial orders* not *linear orders*. In fact, for both T_1 and T_2 there are an infinite number of minimal elements. In particular

- the minimal elements for T_1 are $\{(x,y) : x \leq 0\}$, and
- the minimal elements for T_2 are $\{(x,y) : y < 0\}$.

Recall that the definition of a transition invariant, Definition 7.2, allows partial orders. We see here that this is useful.

10 How Much Ramsey Theory Do We Need?

As mentioned before Podelski and Rybalchenko [36] noted that the proofs of Theorems 5.4, 5.5, and 5.6 do not need the strength of the full Ramsey's Theorem. In the proofs of these theorems the coloring is transitive.

Def 10.1 A coloring of the edges of K_n or K_{N} is *transitive* if, for every $i < j < k$, if $COL(i,j) = COL(j,k)$ then both equal $COL(i,k)$.

Def 10.2 Let $c, n \geq 1$. Let G be K_n or K_{N}. Let COL be a c-coloring of the edges of G. A set of vertices V is a *monochromatic increasing path with respect to COL* if $V = \{v_1 < v_2 < \cdots\}$ and

$$COL(v_1, v_2) = COL(v_2, v_3) = \cdots .$$

(If $G = K_n$ then the $\cdots$ stop at some $k \leq n$.) We will drop the *with respect to COL* if the coloring is understood. We will abbreviate *monochromatic increasing path* by *MIP* from now on.

Here is the theorem we really need. We will refer to it as *the Transitive Ramsey's Theorem*.

Theorem 10.3 *Let $c \geq 1$. For every transitive c-coloring of K_{N} there exists an infinite MIP.*

The Transitive Ramsey Theorem is weaker than Ramsey's Theorem. We show this in three different ways: (1) Reverse Mathematics, (2) Computable Mathematics, (3) Finitary Version.

Def 10.4

1. For all $c \geq 1$ let $RT(c)$ be Ramsey's theorem for c colors.
2. Let RT be $(\forall c)[RT(c)]$.
3. For all $c \geq 1$ let $TRT(c)$ be the Transitive Ramsey's theorem for c colors.
4. Let TRT be $(\forall c)[TRT(c)]$. (This is the theorem that we really need.)

Reverse Mathematics: Reverse Mathematics [42] looks at exactly what strength of axioms is needed to prove results in mathematics. A weak axiom system called RCA_0 (Recursive Comprehension Axiom) is at the base. Intuitively a statement proven in RCA_0 is proven constructively.

Notation 10.5

Let A and B be statements.

- $A \to B$ means that one can prove B from A in RCA_0.
- $A \equiv B$ means that $A \to B$ and $B \to A$.
- $A \not\to B$ means that, only using the axioms in RCA_0, one cannot prove B from A. It may still be the case that A implies B but proving this will require nonconstructive techniques.

The following are known. Items 1 and 2 indicate that the proof-theoretic complexity of RT is greater than that of TRT.

1. $RT \to TRT$. The usual reasoning for this can easily be carried out in RCA_0.
2. Hirschfeldt and Shore [22] have shown that $TRT \not\to RT$.
3. For all c, $RT(2) \equiv RT(c)$. The usual reasoning for this can easily be carried out in RCA_0. Note how this contrasts to the next item.
4. Cholak, Jockusch, and Slaman [8] showed that $RT(2) \not\to (\forall c)[RT(c)]$.

The proof of Theorem 5.4 showed that, over RCA_0,

$$TRT(3) \to \text{Program 4 terminates.}$$

Does the following hold over RCA_0?

$$\text{Program 4 terminates} \to TRT(3).$$

We do not know.

In the spirit of the reverse mathematics program we ask the following: For each c is there a program P_c such that the following holds over RCA_0?

$$P \text{ terminates} \iff TRT(c).$$

The following is open: for which $i, j \geq 2$ does $TRT(i) \to TRT(j)$?

Computable Mathematics: Computable Mathematics [16] looks at theorems in mathematics that are proven non-effectively and questions if there is an effective (that is computable) proof. The answer is usually no. Then the question arises as to how noneffective the proof is. Ramsey's Theorem and the Transitive Ramsey's Theorem have been studied and compared in this light [18, 22, 23, 24, 41].

Def 10.6 Let $M_1^{(\cdots)}, M_2^{(\cdots)}, \ldots$ be a standard list of oracle Turing Machines.

1. If A is a set then $A' = \{e : M_e^A(e) \downarrow\}$. This is also called *the Halting problem relative to A*. Note that $\emptyset' = HALT$.

2. A set A is called *low* if $A' \leq_T HALT$. Note that decidable sets are low. It is known that there are undecidable sets that are low; however, they have some of the properties of decidable sets.

3. We define the levels of the arithmetic hierarchy.

   - A set is in Σ_0 and Π_0 if it is decidable.
   - Assume $n \geq 1$. A set A is in Σ_n if there exists a set $B \subseteq \mathsf{N} \times \mathsf{N}$ that is in Π_{n-1} such that
     $$A = \{x : (\exists y)[(x, y) \in B]\}.$$
   - Assume $n \geq 1$. A set A is in Π_n if $\overline{A}$ is in Σ_n.
   - A set is in the *Arithmetic hierarchy* if it is in Σ_n or Π_n for some n.

The following are known. Items 1 and 3 indicate that the Turing degree of the infinite homogenous set induced by a coloring is greater than the Turing degree of the infinite homogenous set induced by a transitive coloring.

1. Jockusch [24] has shown that there exists a computable 2-coloring of the edges of K_{N} such that, for all infinite homogeneous sets H, H is not computable in the halting set.

2. Jockusch [24] has shown that for every computable 2-coloring of the edges of K_{N} there exists an infinite homogeneous sets $H \in \Pi_2$.

3. For all c, for every computable transitive c-coloring of the edges of K_{N}, there exists an infinite MIP P that is computable in the halting set.

4. There exists a computable transitive 2-coloring of the edges of K_{N} with no computable infinite MIP .

5. Hirschfeldt and Shore [22] have shown that there exists a computable transitive 2-coloring of the edges of K_{N} with no infinite low infinite MIP .

Finitary Version: There are finite versions of both Ramsey's Theorem and the Transitive Ramsey's Theorem. The finitary version of the Transitive Ramsey's Theorem yields better upper bounds.

Notation 10.7 Let $c, k \geq 1$.

1. $R(k,c)$ is the least n such that, for any c-coloring of the edges of K_n, there exists a homogeneous set of size k.

2. $TRT(k,c)$ is the least n such that, for any transitive c-coloring of the edges of K_n, there exists a MIP of length k.

It is not obvious that $R(k,c)$ and $TRT(k,c)$ exist; however, they do.
The following is well known [19, 20, 28].

Theorem 10.8 *For all $k, c \geq 1$, $c^{k/2} \leq R(k,c) \leq c^{ck-c+1}$,*

Improving the upper and lower bounds on the $R(k,c)$ (often called *the Ramsey Numbers*) is a long standing open problem. The best known asymptotic results for the $c = 2$ case are by Conlon [10]. For some exact values see Radziszowski's dynamic survey [37].

The following theorem is easy to prove; however, neither the statement, nor the proof, seem to be in the literature. We provide a proof for completeness.

Theorem 10.9 *For all $k, c \geq 1$ $TRT(k,c) = (k-1)^c + 1$.*

Proof:
I) $TRT(k,c) \leq (k-1)^c + 1$.

Let $n = (k-1)^c + 1$. Assume, by way of contradiction, that there is transitive c-coloring of the edges of K_n that has no MIP of length k.

We define a map from $\{1, \ldots, n\}$ to $\{1, \ldots, k-1\}^c$ as follows: Map x to the the vector $(a_1, \ldots, a_c)$ such that the longest mono path of color i that ends at x has length a_i. Since there are no MIP 's of length k the image is a subset of $\{1, \ldots, k-1\}^c$.

It is easy to show that this map is 1-1. Since $n > (k-1)^c$ this is a contradiction.

2) $TRT(k,c) \geq (k-1)^c + 1$.

Fix $k \geq 1$. We show by induction on c, that, for all $c \geq 1$, there exists a transitive coloring of the edges of $K_{(k-1)^c}$ that has no MIP of length k.
Base Case: $c = 1$. We color the edges of K_{k-1} all RED. Clearly there is no MIP of length k.

Induction Step: Assume there is a transitive $(c-1)$-coloring COL of the edges of $K_{(k-1)^{c-1}}$ that has no homogeneous set of size k. Assume that RED is not used. Replace every vertex with a copy of K_{k-1}. Color edges between vertices in different groups as they were colored by COL. Color edges within a group RED. It is easy to see that this produces a transitive c-coloring of the edges of and that there are no MIP of length k. ∎

Note 10.10 Erdös and Szekeres [15] showed the following:

- For all k, for all sequences of distinct reals of length $(k-1)^2 + 1$, there is either an increasing monotone subsequence of length k or a decreasing monotone subsequence of length k.

- For all k, there exists a sequences of distinct reals of length $(k-1)^2$ with neither an increasing monotone subsequence of length k or a decreasing monotone subsequence of length k.

This is equivalent to the $c = 2$ case of Theorem 10.9. For six different proofs see Steele's article [43]. Our proof of Theorem 10.9 was modeled after Hammersley's [21] proof of the upper bound and Erdös-Szekeres's proof of the lower bound.

If c is small then $TRT(k,c)$ is substantially smaller than $R(k,c)$. This indicates that the Transitive Ramsey's Theorem is weaker than Ramsey's Theorem.

11 Solving Subcases of the Termination Problem

The problem of determining if a program is terminating is unsolvable. This problem is *not* the traditional Halting problem since we allow the program to have a potentially infinite number of user-supplied inputs.

Def 11.1

1. Let $M_1^{(\cdots)}, M_2^{(\cdots)}, \ldots$ be a standard list of oracle Turing Machines. These Turing Machines take input in two ways: (1) the standard way, on a tape, and (2) we interpret the oracle as the user-supplied inputs.

2. If $A \subseteq \mathsf{N}$ and $s \in \mathsf{N}$ then $M_{i,s}^A \downarrow$ means that if you run M_i^A (no input on the tape) it will halt within s steps.

3. Let $M_1^{(\cdots)}, M_2^{(\cdots)}, \ldots$ be a standard list of oracle Turing Machines.

$$TERM = \{i : (\forall A)(\exists s)[M_{i,s}^A \downarrow]\}.$$

Def 11.2

1. $X \in \Pi_1^1$ if there exists an oracle Turing machine $M^{(\cdots)}$ such that

$$X = \{x : (\forall A)(\exists x_1)(\forall x_2)\cdots(Q_n x_n)[M^A(x, x_1, \ldots, x_n) = 1]\}.$$

   (Q_n is a quantifier.)

2. A set X is Π_1^1-complete if $X \in \Pi_1^1$ and, for all $Y \in \Pi_1^1$, $Y \leq_m X$.

The following were proven by Kleene [25, 26] (see also [39]).

Theorem 11.3

1. *$X \in \Pi_1^1$ if there exists an oracle Turing machine $M^{(\cdots)}$ such that*

$$X = \{x : (\forall A)(\exists y)[M^A(x, y) = 1]\}.$$

2. *$TERM$ is Π_1^1-complete.*

3. *If X is Π_1^1-complete then, for all Y in the arithmetic hierarchy, $Y \leq_m X$.*

4. *For all Y in the arithmetic hierarchy $Y \leq_m TERM$. This follows from (2) and (3). (See Definition 10.6 for the definition of the Arithmetic Hierarchy.)*

Hence $TERM$ is much harder than the halting problem. Therefore it will be very interesting to see if some subcases of it are decidable.

Def 11.4 Let $n \in \mathsf{N}$. Let $FUN(n)$ be a set of computable functions from Z^{n+1} to Z^n. Let $m \in \mathsf{N}$. An $(F(n), m))$-*program* is a program of the form of Program 1 where the functions g_i used in Program 1 are all in $FUN(n)$.

Open Question: For which $FUN(n), m$ is the Termination Problem restricted to $(FUN(n), m)$-programs decidable?

We list all results we know. Some are not quite in our framework. Some of the results use the **While** loop condition $Mx \geq b$ where M is a matrix and b is a vector. Such programs can easily be transformed into programs of our form.

1. Tiwari [44] has shown that the following problem is decidable: Given matrices A, B and vector c, all over the rationals, is Program 7 in $TERM$. Note that the user is inputting a real.

Program 7

$x =$ **Input**(R)
while $(Bx > b)$
$\quad x = Ax + c$

2. Braverman [7] has shown that the following problem is decidable: Given matrices A, B_1, B_2 and vectors b_1, b_2, c, all over the rationals, is Program 8 in $TERM$. Note that the user is inputting a real.

Program 8

$x =$ **Input**(R)
while $(B_1x > b_1)$ **and** $(B_2x \geq b_2)$
$\quad x = Ax + c$

3. Ben-Amram, Genaim, and Masud [6] have shown that the following problem is undecidable: Given matrices A_0, A_1, B and vector v all over the integers, and $i \in \mathsf{N}$ does Program 9 terminate.

Program 9

```
x = Input(Z)
while (Bx ≥ b)
        if x[i] ≥ 0
                then x = A_0 x
        else
                x = A_1 x
```

4. Ben-Amram [4] has shown a pair of contrasting results:

   - The termination problem is undecidable for $(FUN(n), m)$-programs where $m = 1$ and $FUN(n)$ is the set of all functions of the form
     $$f(x[1], \ldots, x[n]) = \min\{x[i_1] + c_1, x[i_2] + c_2, \ldots, x[i_k] + c_k\}$$
     where $1 \leq i_1 < \cdots < i_k$ and $c_1, \ldots, c_k \in \mathsf{Z}$.
   - The termination problem is decidable for $(FUN(n), m)$-programs when $m \geq 1$ and $FUN(n)$ is the set of all functions of the form
     $$f(x[1], \ldots, x[n]) = x[i] + c$$
     where $1 \leq i \leq n$ and c$\in \mathsf{Z}$. Note that Program 5 falls into this category.

12 Open Problems

1. For which $(FUN(n), m)$ is the Termination Problem restricted to $(FUN(n), m)$-programs decidable?

2. Find a natural example showing that Theorem 7.3 requires the Full Ramsey Theorem.

3. Prove or disprove that Theorem 7.3 is equivalent to Ramsey's Theorem.

4. Classify more types of Termination problems into the classes Decidable and Undecidable. It would be of interest to get a more refined classification. Some of the undecidable problems may be equivalent to HALT while others may be complete in some level of the arithmetic hierarchy or Π^1_1 complete

5. Prove or disprove the following conjecture: for every c there is a program P_c such that, over RCA_0, $TRT(c) \iff$ every run of Program P_c terminates.

13 Acknowledgments

I would like to thank Daniel Apon, Amir Ben-Amram, Peter Cholak, Byron Cook, Denis Hirschfeldt, Jon Katz, Andreas Podelski, Brian Postow, Andrey Rybalchenko, and Richard Shore for helpful discussions. We would also like to again thank Amir Ben-Amram for patiently explaining to me many subtle points that arose in this paper. We would also like to thank Daniel Apon for a great proofreading job.

A Using Theorem 6.7 and 2×2 Matrices to Prove Termination of Program 5

Def A.1 If $\mathcal{C}$ is a set of square matrices of the same dimension then clos($\mathcal{C}$) is the set of all finite products of elements of $\mathcal{C}$. For example, if $\mathcal{C} = \{C_1, C_2\}$ then $C_1^2 C_2 C_1^3 C_2^{17} \in \text{clos}(C_1, C_2)$.

This section is due to Ben-Amram and is based on a paper of his [4]. He gives an example of a proof of termination of Program 5 where he uses the matrices C_1, C_2 that come out of Program 5 directly (in contrast to our proof in Theorem 8.1 which used 3×3 matrices by introducing $x+y$). Of more interest: there *is* an element of clos(C_1, C_2) that has no negative numbers on the diagonal, namely C_2 itself. Hence we cannot use Theorem 6.4 to prove termination. We can, however, use Theorem 6.7.

Theorem A.2 *Every computation of Program 5 is finite.*

Proof:

The case control=1 is represented by the matrix

$$C_1 = \begin{pmatrix} -1 & 0 \\ \infty & \infty \end{pmatrix}.$$

The case control=2 is represented by the matrix

$$C_2 = \begin{pmatrix} \infty & -2 \\ +1 & \infty \end{pmatrix}.$$

We find a representation of a *superset* of clos(C_1, C_2). Let

$$\mathcal{E} = YZ^a, \qquad \text{where } Y \in \{C_1, C_2, C_1C_2, C_2C_1\} \text{ and } Z = \begin{pmatrix} -1 & \infty \\ \infty & -1 \end{pmatrix}.$$

We show that clos$(C_1, C_2) \subseteq \mathcal{E}$. We will then show that every element of $\mathcal{E} = \text{clos}(\mathcal{E})$ satisfies the premise of Theorem 6.7. We prove this by induction on the number of matrices are multiplied together to form the element of clos(C_1, C_2).

The base case is trivial since clearly $C_1, C_2 \in \mathcal{E}$.

We show the induction step by multiplying each of the four "patterns" in $\mathcal{E}$ *on the left* by each of the matrices C_1, C_2. We use the following identities: $C_1^2 = ZC_1 = C_1Z = C_1C_2$, $C_2^2 = Z$, $ZC_2 = C_2Z$.

1. $C_1(C_1Z^a) = C_1^2Z^a = C_1ZZ^a = C_1Z^{a+1}$
2. $C_2(C_1Z^a) = (C_2C_1)Z^a$
3. $C_1(C_2Z^a) = (C_1C_2)Z^a$
4. $C_2(C_2Z^a) = C_2^2Z^a = ZZ^a = Z^{a+1}$
5. $C_1(C_1C_2Z^a) = C_1^2C_2Z^a = C_1ZC_2Z^a = C_1C_2ZZ^a = (C_1C_2)Z^{a+1}$
6. $C_2(C_1C_2Z^a) = C_2(C_1ZZ^a = (C_2C_1)Z^{a+1}$
7. $C_1(C_2C_1Z^a) = (C_1C_2)C_1Z^a = C_1(ZC_1)Z^a = C_1^2Z^{a+1} = C_1Z^{a+2}$
8. $C_2(C_2C_1Z^a) = ZC_1Z^a = C_1Z^{a+1}$

We have shown that $\mathrm{clos}(C_1, C_2) \subseteq \mathcal{E}$. Now it remains to verify that every matrix represented by $\mathcal{E}$, *or some power thereof*, has a negative integer on the diagonal. Note that in one of the cases, squaring is necessary.

1. $C_1Z^a = \begin{pmatrix} -1-a & -a \\ \infty & \infty \end{pmatrix}$
2. $(C_2Z^a)^2 = C_2^2Z^{2a} = Z^{2a+1}$
3. $C_1C_2Z^a = C_1ZZ^a = C_1Z^{a+1}$
4. $C_2C_1Z^a = \begin{pmatrix} \infty & \infty \\ -3 & -2 \end{pmatrix} Z^a = \begin{pmatrix} \infty & \infty \\ -3 & -2-a \end{pmatrix}$.

We can now apply Theorem 6.7 and we are done. ∎

References

[1] Acl2: Applicative common lisp 2. Used for Modeling, simulation, and inductive reasoning. `http://acl2s.ccs.neu.edu/acl2s/doc/`.

[2] Aprove: Automatic program verification environment. `http://aprove.informatik.rwth-aachen.de/`.

[3] Julia. Helps find bug in programs. `http://julia.scienze.univr.it/`.

[4] A. M. Ben-Amram. Size-change termination with difference constraints. *ACM Trans. Program. Lang. Syst.*, 30(3):1–31, 2008. http://doi.acm.org/10.1145/1353445.1353450.

[5] A. M. Ben-Amram. Size-change termination, monotonicity constraints and ranking functions. *Logical Methods in Computer Science*, 6(3), 2010. http://www2.mta.ac.il/~amirben/papers.html.

[6] A. M. Ben-Amram, S. Genaim, and A. N. Masud. On the termination of integer loops. In *Verification, Model Checking, and Abstract Interpretation, 13th International Conference, VMCAI 2012, Proceedings*, pages 72–87, 2012. http://dx.doi.org/10.1007/978-3-642-27940-9_6. A journal version is under review.

[7] M. Braverman. Termination of integer linear programs. In *Proceedings of the 18th Annual International Conference on Computer Aided Verification* Seattle WA*, LNCS 4144*. Springer, 2006. http://www.cs.toronto.edu/~mbraverm/Pub-all.html.

[8] P. Cholak, C. Jockusch, and T. Slaman. On the strength of ramsey's Theorem for pairs. *Journal of Symbolic Logic*, pages 1–55, 2001. http:www.nd.edu/~cholak/papers/.

[9] M. Codish and C. Taboch. A semantic basis for termination analysis of logic programs. *The Journal of Logic Programming*, 41(1):103–123, 1999. preliminary (conference) version in LNCS 1298 (1997), http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.31.5039.

[10] D. Conlon. A new upper bound for diagonal Ramsey numbers. *Annals of Mathematics*, 2009. http://www.dpmms.cam.ac.uk/~dc340.

[11] B. Cook, A. Podelski, A. Rybalachenko, J. Berdine, A. Gotsman, P. O'Hearn, D. Distefano, and E. Koskinen. Terminator. http://www7.in.tum.de/~rybal/papers/.

[12] B. Cook, A. Podelski, and A. Rybalchenko. Abstraction refinement for termination. In *Proceedings of the 12th Symposium on Static Analysis*, 2005. http://www7.in.tum.de/~rybal/papers/.

[13] B. Cook, A. Podelski, and A. Rybalchenko. Termination proofs for systems code. In *Proceedings of the 27nd Conference on Programming Language design and Implementation*, 2006. http://www7.in.tum.de/~rybal/papers/.

[14] B. Cook, A. Podelski, and A. Rybalchenko. Proving programs terminate. *Communications of the ACM*, 54(5):88, 2011. http://www7.in.tum.de/~rybal/papers/.

[15] P. Erdős and G. Szekeres. A combinatorial problem in geometry. *Compositio Math*, 2:463–470, 1935. http://www.renyi.hu/~p_erodso/1935-01.pdf.

[16] Y. L. Ershov, S. S. Goncharov, A. Nerode, and J. B. Remmel, editors. *Handbook of Recursive Mathematics*. Elsevier North-Holland, Inc., New York, 1998. Comes in two volumes. Volume 1 is Recursive Model Theory, Volume 2 is Recursive Algebra, Analysis, and Combinatorics.

[17] R. Floyd. Assigning meaning to programs. In *PSAM*, 1967. `http://www.cs.virginia.edu/~weimer/2007-615/reading/FloydMeaning.pdf`.

[18] W. Gasarch. A survey of recursive combinatorics. In Ershov, Goncharov, Nerode, and Remmel, editors, *Handbook of Recursive Algebra*. North Holland, 1997. `http://www.cs.umd.edu/~gasarch/papers/papers.html`.

[19] W. Gasarch. Ramsey's theorem on graphs, 2005. `http://www.cs.umd.edu/~gasarch/mathnotes/ramsey.pdf`.

[20] R. Graham, B. Rothschild, and J. Spencer. *Ramsey Theory*. Wiley, 1990.

[21] J. Hammersley. A few seedlings of research. In *Proc. 6th Berkeley Symp. Math. Stat.*, pages 345–394, 1972. `http://projecteuclid.org/DPubS?service=UI&version=1.0&verb=Display&hand%le=euclid.bsmsp/1200514101`.

[22] D. Hirschfeldt and R. Shore. Combinatorial principles weaker than Ramsey's theorem for pairs. *Journal of Symbolic Logic*, 72:171–206, 2007. `http://www.math.cornell.edu/~shore/papers.html`.

[23] T. Hummel. Effective versions of Ramsey's theorem: Avoiding the cone above **0'**. *Journal of Symbolic Logic*, 59(4):682–687, 1994. `http://www.jstor.org/action/showPublication?journalCode=jsymboliclogic`.

[24] C. Jockusch. Ramsey's theorem and recursion theory. *Journal of Symbolic Logic*, 37:268–280, 1972. `http://www/jstor.org/pss/2272972`.

[25] S. Kleene. Hierahies of number theoretic predicates. *Bulletin of the American Mathematical Society*, 61:193–213, 1955. `http://www.ams.org/journals/bull/1955-61-03/home.html`.

[26] S. C. Kleene. *Introduction to Metamathematics*. D. Van Nostrand, Princeton, 1952.

[27] D. Kroening, N. Sharygina, A. Tsitovich, and C. M. Wintersteiger. Loopfrog. `http://www.verify.inf.unisi.ch/loopfrog/termination`.

[28] B. Landman and A. Robertson. *Ramsey Theory on the integers*. AMS, 2004.

[29] C. S. Lee. Ranking functions for size-change termination. *ACM Transactions on Programming Languages and Systems*, 31(3), Apr. 2009. `http://doi.acm.org/10.1145/1498926.1498928`.

[30] C. S. Lee, N. D. Jones, and A. M. Ben-Amram. The size-change principle for program termination. In *Proceedings of the 28nd Symposiusm on Principles of Programming Languages*, 2001. http://dl.acm.org/citation.cfm?doid=360204.360210.

[31] N. Lindenstrauss and Y. Sagiv. Automatic termination analysis of Prolog programs. In L. Naish, editor, *Proceedings of the Fourteenth International Conference on Logic Programming*, pages 64–77, Leuven, Belgium, Jul 1997. MIT Press.

[32] J.-Y. Moyen. Resource control graphs. In *Transactions on Computational Logic*, 2009. http://www-lipn.univ-paris13.fr/~moyen/publications.html.

[33] A. Podelski and A. Rybalchenko. A complete method for the synthesis of linear ranking functions. In *Proceedings of the 5th International Conference on Verification, Model Checking, and Abstract*, 2004. http://www7.in.tum.de/~rybal/papers/.

[34] A. Podelski and A. Rybalchenko. Transition invariants. In *Proceedings of the Nineteenth Annual IEEE Symposium on Logic in Computer Science,* Turku, Finland, 2004. http://www7.in.tum.de/~rybal/papers/.

[35] A. Podelski and A. Rybalchenko. Transition predicate abstraction and fair termination. In *Proceedings of the 32nd Symposiusm on Principles of Programming Languages*, 2005. http://www7.in.tum.de/~rybal/papers/.

[36] A. Podelski and A. Rybalchenko. Transition invariants and transition predicate abstraction for program termination. In P. A. Abdulla and K. R. M. Leino, editors, *TACAS*, volume 6605 of *Lecture Notes in Computer Science*, pages 3–10. Springer, 2011. http://www7.in.tum.de/~rybal/papers/ or http://dx.doi.org/10.1007/978-3-642-19835-9_2.

[37] S. Radziszowski. Small Ramsey numbers. *The electronic journal of combinatorics*, 2011. www.combinatorics.org. A dynamic survey so year is last update.

[38] F. Ramsey. On a problem of formal logic. *Proceedings of the London Mathematical Society*, 30:264–286, 1930. Series 2. Also in the book *Classic Papers in Combinatorics* edited by Gessel and Rota. Also http://www.cs.umd.edu/~gasarch/ramsey/ramsey.html.

[39] H. Rogers, Jr. *Theory of Recursive Functions and Effective Computability.* McGraw Hill, New York, 1967.

[40] A. Rybalchenko. ARMC (abstraction refinement-based model checker. http://www7.in.tum.de/~rybal/papers/.

[41] D. Seetapun and T. A. Slaman. On the strength of Ramsey's Theorem. *Notre Dame Journal of Formal Logic*, 36:570–581, 1995. http://projecteuclid.org/DPubS?service=UI&version=1.0&verb=Display&hand%le=euclid.ndjfl/1040136917.

[42] S. G. Simpson. *Subsystems of Second Order Arithmetic*. Springer-Verlag, 2009. Perspectives in mathematical logic series.

[43] J. Steele. Variations on the monotone subsequence theme of Erdős and Szekeres. In D. A. et al, editor, *Discrete probability and algorithms*, pages 111–131, 1995. `http://www-stat.wharton.upenn.edu/~steele/Publications/`.

[44] A. Tiwari. Termination of linear programs. In *Proceedings of the 16th Annual International Conference on Computer Aided Verification* Boston MA*, LNCS 3114*, volume 3115 of *Lecture Notes in Computer Science*. Springer, 2004. `http://www.csl.sri.com/users/tiwari/html/cav04.html`.