An Application of Ramsey's Theorem to Proving Programs Terminate

Exposition by William Gasarch

1 Introduction

We describe an application of Ramsey's Theorem to proving programs terminate, by which we mean terminate on any input. This paper is self contained; it does not require knowledge of either Ramsey's Theorem or Programming Languages.

Our account is based on the articles of B. Cook, Podelski, and Rybalchenko [11, 26] and Lee, Jones, and Ben-Amram [21]. Many other papers [9, 10, 25, 27, 28] have used the techniques I present here. Termination checkers that use the methods of B. Cook, Podelski, and Rybalchenko include ARMC [31], Loopfrog [19], and Terminator [8]. Earlier Terminator checkers that used methods from [25] are Terminlog [22] and Terminweb [6]. 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 bizarre) inputs that make the program run forever.

In the summary below we refer to Programs 1,2,3 and 4 which appear later in the paper.

1. Section 3: We prove Program 1 terminates. This proof uses a simple well founded ordering. We then state Theorem 3.2 that encapsulates this kind of proof.

2. Section 4: We prove Program 2 terminates. This proof uses a complicated well founded ordering and Theorem 3.2.

3. Section 5: We prove Program 2 terminates using Ramsey's Theorem. We then state Theorems 5.6 and Theorem 5.7 which encapsulates this kind of proof. This proof of termination is easier than the proof is Section 4 in some ways, but more complicated in other ways.

4. Section 6: We prove Program 2 terminates using Theorem 5.6 and hence using Ramsey's Theorem; however, we prove the premise of Theorem 5.6 using matrices. We then state and prove Theorems 6.4 and 6.5 that encapsulates this kind of proof. We discuss how this approach can be made into an algorithm that generates proofs-of-termination in some cases.

5. Section 7: We prove Program 3 terminates using Theorem 5.7 and hence using Ramsey's Theorem; however, we prove the premise of Theorem 5.7 using transition invariants. We then state Theorem 7.4 that encapsulates this kind of proof. It seems difficult to obtain a proof that Program 3 terminates without using Ramsey's Theorem.

Program 0

```
(x,y) = (input(NAT),input(NAT))
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)
```

6. Section 8: We prove Program 3 terminates using Theorem 6.5 and hence using Ramsey's Theorem and matrices. Program 3 has some properties that make this a good illustration.

7. Section 9: We prove Program 4 terminates using Theorem 7.4 and hence Ramsey's Theorem and transition invariants. Program 4 has some properties that make this a good illustration.

8. Section 10: We show that the full strength of Ramsey's Theorem is not needed for the proofs of Theorems 5.5, 5.6,and 5.7 and discuss what is needed.

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

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* $n \in \{1, 2, 3, \ldots, \}$. We use N in prose and NAT in programs.

2. Z is the set of integers, $\{\ldots, -2, -1, -, 1, 2, \ldots\}$. We use Z in prose and INT in programs.

3. Q is the set of rationals. We use Q in prose and RAT in programs.

Notation 2.2 In a program the command
x = input(INT)
means that x gets an integer provided by the user. More generally, if A is any set, then
x = input(A)
means that x gets a value from A provided by the user.

All of the programs we deal with in this paper 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.

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 set of states. (If $(\mathrm{x}_1, \ldots, \mathrm{x}_n)$ 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 subset of S. (I is the set of states that the program could be in initially.)
   - $R \subseteq S \times S$. ($R(s,t)$ 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, $R(s_i, s_{i+1})$.
   - 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. Henceforth we denote this comp. seg.

Example 2.4 Program 0 can be defined by the following.

- $S = \mathsf{Z} \times \mathsf{Z}$.
- $I = \mathsf{N} \times \mathsf{N}$.
- $R = \{(x,y), (x+10, y-1) \mid x, y \geq 1\} \bigcup \{(x,y), (y+17, x-2) \mid x, y \geq 1\}$.

Note 2.5 All programs in this paper will be of the same form as Program 5.

Program 1

```
(x,y,z0 = (input(INT), input(INT), input(INT))
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)
```

3 A Proof of Termination Using a Simple Well Founded Ordering

Consider Program 1. We want to prove that *every* computation of Program 1 is finite. That is, whatever the user inputs, the program will terminate. The key is to find some 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 1 is finite.*

Proof:

Let

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

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

The key to the proof of Theorem 3.1 is

1. We map each state of the program, (x,y,z), to an element of N, denoted $f(\mathrm{x},\mathrm{y},\mathrm{z})$.
2. We show that at every iteration of the **While** loop, f decreases.
3. Once f hits 0 the program must terminate.

There is a more general theorem, due to Floyd [13] lurking here which we state without proof. Our statement uses a different notation than his.

Program 2

```
(w,x,y,z)= (input(INT),input(INT),input(INT),input(INT))
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
```

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.*

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 the minimum element of P, then the program terminates.*

Then any computation of $PROG$ is finite.

4 A Proof of Termination Using an Ordering on 4-Tuples

Consider Program 2. We want to prove that *every* computation of Program 2 is finite. That is, whatever the user inputs, the program will terminate. The key is to find some quantity that, during every iteration of the **While** Loop, decreases. None of w,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

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

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 2 is finite.*

Proof:

Let

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

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 $f(\mathrm{w}, \mathrm{x}, \mathrm{y}, \mathrm{z})$ decreases.

Proof of Claim 1:

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

1. control=1: w decreases by 1 and x's value changes (it may increase a lot). Since the order is lexicographic, and w is the first coordinate, and x is the second coordinate, if w decreases by 1 then the tuple decreases no matter how x changes.

2. control=2: x decreases by 1 and y's value changes (it may increase a lot). This case is similar to the control=1 case.

3. control=3: y decreases by 1 and z's value changes (it may increase a lot). This case is similar to the control=1 case.

End of Proof of Claim 1

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

Proof of Claim 2:

If $f(\mathrm{w}, \mathrm{x}, \mathrm{y}, \mathrm{z}) = (0, 0, 0, 0)$ 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 2 terminates. ∎

5 A Proof of Termination Using Ramsey's Theorem

In the proof of Theorem 4.3 we showed that during every single step of Program 2 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 a complicated ordering.

In this chapter we will prove that Program 2 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 comp. seg, one of those quantities decreases.

We will use the infinite Ramsey's Theorem [30] (see also [14, 15, 20]) 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.

Ramsey's Theorem:

Theorem 5.3 *Let $c \geq 1$. Every c-coloring of the the edges of K_{N} has an infinite homogeneous set.*

Note 5.4 The term *Ramsey's Theorem* often refers to the version on hypergraphs. In this paper we take it to mean Theorem 5.3 which is just for graphs.

Theorem 5.5 *Every computation of Program 2 is finite.*

Proof:

We first show that for every finite comp. seg. one of w,x,y,z 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.
2. If control=2 ever occurs in the segment then x decreases. Since control=0 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.
3. If control=3 is the only case that occurs in the segment then y decreases.

We show Program 2 terminates. Assume, by way of contradiction, that there is an infinite computation. Let this computation be

$(\mathrm{w}_1, \mathrm{x}_1, \mathrm{y}_1, \mathrm{z}_1), (\mathrm{w}_2, \mathrm{x}_2, \mathrm{y}_2, \mathrm{z}_2), \ldots$

Since in every comp. seg. one of w, x, y, z decrease we have that, for all $i < j$, either $\mathrm{w}_i > \mathrm{w}_j$ or $\mathrm{x}_i > \mathrm{x}_j$ or $\mathrm{y}_i > \mathrm{y}_j$ or $\mathrm{z}_i > \mathrm{z}_j$. We use this to create a coloring of the edges of the K_{N}. Our colors are W, X, Y, Z. In the coloring below each case assumes that the cases above it did not occur.

$$COL(i,j) = \begin{cases} W \text{ if } \mathrm{w}_i > \mathrm{w}_j; \\ X \text{ if } \mathrm{x}_i > \mathrm{x}_j; \\ Y \text{ if } \mathrm{y}_i > \mathrm{y}_j; \\ Z \text{ if } \mathrm{z}_i > \mathrm{z}_j. \end{cases} \tag{3}$$

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 5.7.)

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

$$\mathrm{w}_{i_1} > \mathrm{w}_{i_2} > \mathrm{w}_{i_3} > \cdots .$$

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

The ideas in the proof of Theorem 5.5 are from Theorem 1 of [26], though similar ideas were in [21]. In Theorem 5.5 we showed that, for any comp. seg, one of the variables decreased. If some function of the variables decreased, this would have sufficed. The next two theorems, which are subcases of Theorem 1 of [26], states this.

Theorem 5.6 *Let $PROG = (S, I, R)$ be a program of the form of Program 5. Note that the variables are x[1],...,x[n]. If for all comp. seg. $s_1, \ldots, s_n$ 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 5.6, captures this.

Theorem 5.7 *Let $PROG = (S, I, R)$ be a program. Assume that there exists well founded orderings $(P_1, <_1), \ldots, (P_m, <_m)$ and functions $f_1, \ldots, f_m$ such that $f_i : S \to P_i$. Assume the following.*

1. *For all comp. seg. $s_1, \ldots, s_n$ there exists i such that $f_i(s_n) <_i f_i(s_1)$.*

2. *If the program is in a state s such that, for some k, $f_k(s)$ is the minimum element of P_k, then the program terminates.*

Then any computation of PROG is finite.

Note 5.8 The proofs of Theorems 5.6 and 5.7 do not need the full strength of Ramsey's Theorem. We will comment on this in Section 10.

6 Proof of Termination Using Matrices and Ramsey's Theorem

Part of the proof of Theorem 5.5 involved showing that, for any finite comp. seg. of Program 2, one of w,x,y,z decreases. Can such proofs be automated? Lee, Jones, and Ben-Amram [21] developed a way to partially automating such proofs. They use matrices. The other part of the proof of Theorem 5.5 used Ramsey's Theorem. Hence they prove programs terminate using matrices and Ramsey's Theorem.

We use their techniques to give a proof that Program 2 terminates. We will then discuss their general technique. Cook, Podelski, Rybalchenko have also developed a way to partially automate the such proofs. We discuss this in Section 5.

Let P be a program with variables x[1],...,x[n] and control takes values in $\{1, ..., m\}$. Let $1 \leq k \leq m$. Let x[1],...,x[n] be the values of the variables before the control=k code executes and let x[1]',...,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 in a nonstandard way) 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 2. The matrix for control=1 , which we denote A, is

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

The matrix for control=2 is, which we denote B, is

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

The matrix for control=3 is, which we denote C, is

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

We want to define matrix multiplication such that if X_a is the matrix for control=a and X_b is the matrix for control=b then X_aX_b is the matrix for what happens if first the control=a code is executed and then the control=b code is executed.

1. A decrease of L_1 (hence the entry $-L_1$) followed by a decrease of L_2 (hence the entry $-L_2$) should yield a decease of $L_1 + L_2$ (hence the entry $-(L_1 + L_2)$. Hence instead of multiplying $-L_1$ by $-L_2$ we add them.

2. If we do not know how $x[i]'$ and $x[j]$ relate (hence an entry of ∞) then even if we know how $x[i]''$ and $x[j]'$ relate, we do not know how $x[i]''$ and $x[i]$ relate. Hence instead of multiplying ∞ by some element of $\mathsf{Z} \cup \{\infty\}$ we add them to get ∞.

3. If we do not know how $x[i]''$ and $x[j]'$ relate (hence an entry of ∞) then even if we know how $x[i]$ and $x[j]'$ relate, we do not know how $x[i]''$ and $x[i]$ relate. Hence instead of multiplying some element of $\mathsf{Z} \cup \{\infty\}$ we add them to get ∞.

4. Using items 1,2 and 3 we will always add rather than multiply.

5. Lets call the variables x[1],. . .,x[n]. It is possible that from the first matrix we know x[i]≥x[k]'+L_1'. and from the second one we know x[k]'≥x[j]"+L_2". Then we know x[i]≥x[j]"+$(L_1 + L_2)$". The way to achieve this we use min over k instead of addition.

6. In summary if X_a has entries $a[i, j]$ and X_b has entries $b[i, j]$ X_aY_b has in its (i,j) entry

$$\min_k \{a[i, k] + a[k, j]\}.$$

The following theorem, from [21], we leave for the reader.

Lemma 6.2 *Let $PROG_a$ and $PROG_b$ be programs that use the variables x[1],. . . ,x[n]. (We think of $PROG_a$ and $PROG_b$ as being what happens in the various control cases.) Let X_a be the matrix that represent what is known whenever $PROG_a$ is executed. Let X_b bt the matrix that represent what is known whenever $PROG_b$ is executed. Then the matrix produce X_aX_b as defined above represents what is known when $PROG_a$ and then $PROG_b$ are executed.*

We return to our example.

Theorem 6.3 *Every computation of Program 2 is finite.*

Proof:

We show that the premises of Theorem 5.6 hold. Item 1 is easily proven directly. Items 2,3,4,5,6,7 are easily proven by induction on the number of matrices being multiplied.

1. $AB = BA$, $AC = CA$, $BC = CB$.
2. For all $i \geq 1$ A^i is
$$\begin{pmatrix} -i & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & 0 \end{pmatrix}$$
3. For all $j \geq 1$ B^j is
$$\begin{pmatrix} 0 & \infty & \infty & \infty \\ \infty & -j & \infty & \infty \\ \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & 0 \end{pmatrix}$$
4. For all $k \geq 1$ C^k is
$$\begin{pmatrix} 0 & \infty & \infty & \infty \\ \infty & 0 & \infty & \infty \\ \infty & \infty & -k & \infty \\ \infty & \infty & \infty & \infty \end{pmatrix}$$
5. For all $i, k \geq 1$ A^iC^k is
$$\begin{pmatrix} -i & \infty & \infty & \infty \\ \infty & 0 & \infty & \infty \\ \infty & \infty & -k & \infty \\ \infty & \infty & \infty & \infty \end{pmatrix}$$
6. For all $j, k \geq 1$ B^jC^k is
$$\begin{pmatrix} 0 & \infty & \infty & \infty \\ \infty & -j & \infty & \infty \\ \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty \end{pmatrix}$$
7. For $i, j, k \geq 1$ and $k \geq 0$, $A^iB^jC^k =$
$$\begin{pmatrix} -i & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & 0 \end{pmatrix}$$

   Note that the result does not depend on j or k.

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 A, B, C. Since the multiplication of these matrices is commutative we need only concern ourselves with $A^i B^j C^k$ for $i, j, k \in \mathsf{N}$. In all of the cases below $i, j, k \geq 1$.

1. A^i: w decreases.
2. B^j: x decreases.
3. C^k: y decreases.
4. $A^i B^j$: x decreases.
5. $A^i C^k$: Both w and y decrease.
6. $B^j C^k$: x decreases.
7. $A^i B^j C^k$: w decreases.

Hence, for any any comp. seg. $s_1, \ldots, s_n$ of Program 2 either w,x,y, or z decreases. Hence by Theorem 5.6 Program 2 terminates.
∎

The proof technique we used above is quite general. Basically you form the matrices and see if all products of them has a negative number on the diagonal. Lee, Jones, and Ben-Amram [21](Theorem 4) have essentially shown the following:

Theorem 6.4 *Let $PROG = (S, I, R)$ be a program in the form of Program 5. Let $A_1, A_2, \ldots, A_m$ be the matrices the $n \times n$ matrices associated to control=1, ..., control=m cases. If every product of the A_i's yields a matrix with a negative integer on the diagonal then the program terminates.*

Proof: Consider comp. seg. $s_1, \ldots, s_n$. Let the corresponding matrices be $A_{i_1}, \ldots, A_{i_n}$. By the premise the product of these matrices has a negative integer on the diagonal. Hence some variable decreases. By Theorem 5.6 the program terminates. ∎

Theorem 6.4 leads to the following potential algorithm to test if programs of the type we have been considering halt.

1. Input Program P.
2. Form matrices for all the cases of control.
3. Determine if it is the case that all possible products of these matrices have a negative number on the diagonal. (This step may be difficult.)
4. If so then output YES the program terminates.

5. If not the then output DON"T KNOW.

This algorithm can be used to prove that some programs terminate but 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.

Theorem 6.5 *Let $PROG = (S, I, R)$ be a program in the form of Program 5. Let $f_1, \ldots, f_p$ be functions of the n variables. Let $A_1, A_2, \ldots, A_m$ be the $p \times p$ matrices that describe 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. Assume that when one of the f_i is ≤ 0 then the* **While** *loop condition does not hold so the program stops. If every product of the A_i's yields a matrix with a negative integer on the diagonal then the program terminates.*

Theorem 6.5 also leads to the a potential 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.

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

Theorem 6.6 *Let $PROG$, $f_1, \ldots, f_p$, $A_1, \ldots, A_m$ be as in Theorem 6.5. Further assume that each column of each A_i has at most one non-infinity value. If every product of the A_i's is such that some power of it has a negative on the diagonal then the program terminates.*

Proof:

FILL IN LATER

∎

The condition on the columns turns out to not be necessary. Jean-Yves Moyen has shown the following (you need to combine [23] and [24].

Theorem 6.7 *Let $PROG$, $f_1, \ldots, f_p$, $A_1, \ldots, A_m$ be as in Theorem 6.5. If every product of the A_i's is such that some power of it has a negative on the diagonal then the program terminates.*

Proof:

IF FIGURE IT OUT FILL IN LATER

∎

The condition on the columns turns out to not be necessary. Jean-Yves Moyen has shown the following (you need to combine [23] and [24].

Program 3

```
(x,y) = (input(INT),input(INT))
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)
```

7 A Proof of Termination Where Ramsey's Theorem Makes it Much Easier

We proved that Program 2 terminates in three different ways. The proof in Theorem 4.3 used a complicated well founded order; however, the proof only had to deal with what happened during one step of Program 2. The proofs in Theorem 5.5 and 6.3 used four simple well founded orders and Ramsey's Theorem; however, the proofs had to deal with *any* comp. seg. of Program 2. 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 [26] of a program (Program 3 in the paper you are reading) where the proof of termination using Ramsey's Theorem is easy (Podelski and Rybalchenko found it by hand and later their termination checker found it automatically) but the proof of termination using a well founded ordering seems difficult (we have not been able to find one).

We want to prove that *every* computation of Program 3 is finite. That is, whatever the user inputs, the program will terminate. The key is to find some set of quantities such that, for every comp. seg, one of them decreases. We will use x,y, and x+y.

Theorem 7.1 *Every computation of Program 3 is finite.*

Proof:

We assume that the comp. seg. enters the **While** loop, else the program has already terminated.

We show that both premises of Theorem 5.7 hold with $P_1 = P_2 = P_3 = \mathsf{N}$, $f_1(\mathrm{x},\mathrm{y}) = \mathrm{x}$, $f_2(\mathrm{x},\mathrm{y}) = \mathrm{y}$, and $f_3(\mathrm{x},\mathrm{y}) = \mathrm{x}+\mathrm{y}$. 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 comp. seg, one of x,y,x+y decreases.

Proof of Claim 1:

We want to prove that, for all $n \geq 2$, for all comp. segs. of length n

$$(\mathrm{x}_1,\mathrm{y}_1),(\mathrm{x}_2,\mathrm{y}_2),\ldots,(\mathrm{x}_n,\mathrm{y}_n),$$

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 comp. segs. 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.

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

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

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

   so (1) holds.

   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 $y_n \leq x_1 + 1$

   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.

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 $y_n < y_1$

   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.

   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 $y_n < y_1$

   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.

   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 comp. seg. 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 5.7 is satisfied. Hence Program 3 terminates.

∎

In the proof of Theorem 7.1 we only needed to show that, for any comp. seg, either x or y decreased.

1. We actually showed that either x,y, or x+y decreased. Clearly this is true iff one of x or y decreased. However, if we had tried to show that one of x,y decreased directly the invariant would have to be more complicated.

2. We actually showed that, for any comp. seg, the invariant held. This is a case of strengthening the induction hypothesis. This is the key to the proof. We show how it can be generalized below.

We can state the invariant differently. Consider the following four orderings on $\mathsf{N} \times \mathsf{N}$ (we write the ordered pairs $((x,y),(x',y))$ to indicate $(x,y) > (x',y')$).

$$\begin{array}{rl} T_1 & = \{((x,y),(x',y')) \mid \mathrm{x} > 0 \text{ and } \mathrm{y} > 0 \text{ and } \mathrm{x}' < \mathrm{x} \text{ and } \mathrm{y}' \leq \mathrm{x}\} \\ T_2 & = \{((x,y),(x',y')) \mid \mathrm{x} > 0 \text{ and } \mathrm{y} > 0 \text{ and } \mathrm{x}' < \mathrm{y}-1 \text{ and } \mathrm{y}' \leq \mathrm{x}+1\} \\ T_3 & = \{((x,y),(x',y')) \mid \mathrm{x} > 0 \text{ and } \mathrm{y} > 0 \text{ and } \mathrm{x}' < \mathrm{y}-1 \text{ and } \mathrm{y}' < \mathrm{y}\} \\ T_4 & = \{((x,y),(x',y')) \mid \mathrm{x} > 0 \text{ and } \mathrm{y} > 0 \text{ and } \mathrm{x}' < \mathrm{x} \text{ and } \mathrm{y}' < \mathrm{y}\} \\ T & = T_1 \cup T_2 \cup T_3 \cup T_4 \end{array}$$

Note that (1) each T_i is well founded, and (2) for any comp. seg. $(\mathrm{x}_1,\mathrm{y}_1),(\mathrm{x}_2,\mathrm{y}_2),\ldots,(\mathrm{x}_n,\mathrm{y}_n)$ we have $((\mathrm{x}_1,\mathrm{y}_1),(\mathrm{x}_n,\mathrm{y}_n)) \in T$.

It is easy to see that these properties of T are all we needed in the proof. This is Theorem 1 of [26] which we state and prove. This is the only theorem using Ramsey's theorem in this paper that uses the full strength of Ramsey's Theorem. We discuss this more in Section 10.

Def 7.2 An ordering T is well founded if every set has a minimal element. Note that any infinite sequence must have a minimum element.

Def 7.3 Let $PROG = (S, I, R)$ be a program.

1. An ordering T such that $S \times S \subseteq T$ is a *transition invariant* if for any comp. seg. $s_1,\ldots,s_n$ we have $(s_1,s_n) \in T$

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

Proof: Let $T = T_1 \cup \cdots \cup T_k$ be the disjunctive well-founded transition invariant for $PROG$.

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 be

$COL(i, j) =$ the least L such that $(s_i, s_j) \in T_L$.

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 T_c. Then

$$s_{i_1} <_c s_{i_2} <_c < \cdots <$$

This contradicts $<_c$ being well founded. ∎

Finding an appropriate T is the key to the proofs of termination for the termination checkers Loopfrog [19], and Terminator [8].

FILL IN LATER- DISCUSSION OF NOT USING TRAN RAMSEY, AND EXAMPLES. LATER ON SPECULATE IF EQUIV TO FULL RAMSEY OVER RCA_0.

8 Another Proof of Termination Using Matrices and Ramsey's Theorem

Theorem 8.1 *Every computation of Program 3 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 3 corresponds to the following matrix which we denote A.

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

The control=2 case of Program 3 corresponds to the following matrix which we denote B.

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

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 $i \geq 1$ A^i is

$$\begin{pmatrix} -i & -i+1 & -i+2 \\ \infty & \infty & \infty \\ \infty & \infty & \infty \end{pmatrix}$$

2. For all $i \geq 1$, j odd, $j = 2k-1$, B^j is

$$\begin{pmatrix} -k & \infty & \infty \\ \infty & -k & \infty \\ \infty & \infty & -2k \end{pmatrix}$$

3. For all $j \geq 2$, j even, $j = 2k$, B^j is

$$\begin{pmatrix} \infty & -k+1 & \infty \\ -k-2 & \infty & \infty \\ \infty & \infty & -2k-1 \end{pmatrix}$$

4. If $i, j \geq 1$ then $A^i B^j$ is

$$\begin{pmatrix} -(i+\lceil j/2 \rceil & -(i-1+\lfloor j/2 \rfloor & -(j-1) \\ \infty & \infty & \infty \\ \infty & \infty & \infty \end{pmatrix}$$

5. If $i, j \geq 1$, i is odd, then $B^i A^j$ is

$$\begin{pmatrix} \infty & \infty & \infty \\ -(\lfloor i/2 \rfloor + j + 2 & -(\lfloor i/2 \rfloor + j + 1 & -(\lfloor i/2 \rfloor + j \\ \infty & \infty & \infty \end{pmatrix}$$

6. If $i, j \geq 1$, i is even, then $B^i A^j$ is

$$\begin{pmatrix} -(i/2)+j & -(i/2)+j-1 & -\lfloor i/2 \rfloor + j - 2 \\ \infty & \infty & \infty \\ \infty & \infty & \infty \end{pmatrix}$$

We use this information to formulate a lemma.

Def 8.2 If we put < 0 (≤ 0) in an entry of a matrix it means any integer that is less than 0 (less than or equal to 0).

Lemma 8.3 *For all $n \geq 2$, any product of n matrices all of which are A's and B'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.
5. $$\begin{pmatrix} \infty & < 0 & \infty \\ < 0 & \infty & \infty \\ \infty & \infty & < 0 \end{pmatrix}$$

This can be proved easily by induction on n. ∎

Note 8.4 One can show that every computation of Program 3 terminates using 2×2 matrices. Amir Ben-Amram has done this and has allowed us to place his proof in the appendix of this paper.

Theorem 8.5 *Every computation of Program 4 is finite.*

Proof: By Lemma 8.3 every finite product of the matrices A and B has a negative integer on the diagonal. Hence by Theorem 6.5, every computation of Program 4 is finite. ∎

9 Another Example

We want to prove that *any* run of Program 4 will terminate. Intuitively this is easy: eventually y is negative and after that point x will steadily decrease until $x < 0$. But this would be hard for a termination checker since x might increases for a very long time. Instead we need to find the right disjunctive well-founded transition invariant.

Theorem 9.1 *Every run of Program 4 terminates.*

Program 4

```
(x,y) = (input(INT),input(INT))
While x>0
                    (x,y) = (x+y,y−1)
```

Proof: We define orderings T_1 and T_2 (we write it as $>$ instead of $<$) (we write the ordered pairs $((x,y),(x',y))$ to indicate $(x,y) > (x',y')$).

$$T_1 = \{((\mathrm{x},\mathrm{y}),(\mathrm{x}',\mathrm{y}')) \mid \mathrm{x} > 0 \text{ and } \mathrm{x}' < \mathrm{x}\}$$

$$T_2 = \{((\mathrm{x},\mathrm{y}),(\mathrm{x}',\mathrm{y}')) \mid \mathrm{y} \geq 0 \text{ and } \mathrm{y}' < \mathrm{y}\}.$$

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 comp. segs. of length n

$$(\mathrm{x}_1,\mathrm{y}_1),(\mathrm{x}_2,\mathrm{y}_2),\ldots,(\mathrm{x}_n,\mathrm{y}_n)$$

either T_1 or T_2 holds.

We prove this by induction on n.

We will assume that the comp. seg. 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 $\mathrm{y}_1 \geq 0$ then since $\mathrm{y}_1 > \mathrm{y}_1 - 1 = \mathrm{y}_2$ T_2 holds independent of what $\mathrm{x}_1, \mathrm{x}_2$ are.

If $\mathrm{y}_1 < 0$ then $\mathrm{x}_1 > \mathrm{x}_1 + \mathrm{y}_1 = \mathrm{x}_2$. Since $\mathrm{x}_1 > 0$, T_1 holds.

Induction Step: There are four cases based on (1) $\mathrm{y} \leq 0$ or $\mathrm{y} > 0$, and (2) T_1 or T_2 holds between $(\mathrm{x}_1,\mathrm{y}_1)$ and $(\mathrm{x}_n,\mathrm{y}_n)$. We omit details. ∎

Note 9.2 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 minimum elements. In particular

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

Hence the definition of Transitive invariant that allows partial orders is useful.

10 What Do We Need?

Podelski and Rybalchenko [28] noted that the proofs of Theorems 5.5, 5.6,and 5.7 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 $COL(i,k)$.

Also note that the proofs of Theorems 5.5, 5.6,and 5.7. did not need a homogeneous set; all they need is a monochromatic increasing path.

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.*

Note 10.4 Replacing the premise *any c-coloring* with *any transitive c-coloring* does indeed make the Transitive Ramsey's Theorem weaker than Ramsey's Theorem. However, replacing the conclusion *an infinite homogeneous set* with *an infinite mip* does not change the theorem—it is easy to see that, for any transitive c-colorings of the edges of K_{N} there is an infinite homogeneous set iff there is an infinite mip.

Is the Transitive Ramsey's Theorem actually weaker than Ramsey's Theorem? Yes, and in three different ways: (1) Reverse Mathematics, (2) Computable Mathematics, (3) Finitary Version.

Def 10.5

1. For all $c \geq 1$ let $RT(c)$ be Ramsey's theorem for c colors.
2. RT is $(\forall c)[RT(c)]$ which is the usual Ramsey's theorem.
3. For all $c \geq 1$ let $TRT(c)$ be the Transitive Ramsey's theorem for c colors.

4. TRT is $(\forall c)[TRT(c)]$ which is the usual Transitive Ramsey's theorem. This is the theorem that we really need.

Reverse Mathematics: Reverse Mathematics [32] 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. The statement $A \not\rightarrow B$ ($A \rightarrow B$; $A \equiv B$) means that, even allowing the use of the axioms in RCA_0, one cannot prove B from A (one can prove B from A; one can prove B from A, and A from B). The following are known and (items 1 and 2) indicate that the proof-theoretic complexity of RT is greater than that of TRT.

1. $RT \rightarrow TRT$. The usual reasoning for this can easily be carried out in RCA_0.
2. Hirschfeldt and Shore [17] have shown that $TRT \not\rightarrow RT$.
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 showed that $RT(2) \not\rightarrow (\forall c)[RT(c)]$.

For Program X and programs like it we know that there is some c such that

$$TR(c) \rightarrow \text{Program X terminates.}$$

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

$$P \text{ terminates } \rightarrow TR(c).$$

Note that the following is open: for which $i, j \geq 2$ do we have $i, j \geq 2$ if $TRT(i) \rightarrow TRT(j)$? Other variants of Ramsey Theory can also be explored.

Computable Mathematics: Computable Mathematics 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 compared in this light. The following are known and (items 1 and 2) indicate that the complexity of H is larger than the complexity of P.

1. Jockusch [18] has shown there exists a computable 2-coloring of the edges of K_{N} such that, for all homogeneous sets H, H is not computable in the halting set.
2. 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. This is folklore.
3. There exists a computable transitive 2-coloring of the edges of K_{N} with no computable infinite mip. This is folklore.

4. Hirschfeldt and Shore [17] have shown that there exists a computable transitive 2-coloring of the edges of K_{N} with no 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 10.6 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. $TR(k, c)$ 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 $TR(k, c)$ exist; however, they do.
The following is well known [14, 15, 20].

Theorem 10.7 *For all* $k, c \geq 1$ $R(k, c) \leq c^{ck-c+1}$,

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 [7]. For some exact values see Radziszowski dynamic survey [29].

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

Theorem 10.8 *For all* $k, c \geq 1$ $TR(k, c) = (k-1)^c + 1$.

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

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

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 10.9 Erdös and Szekeres [12] 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 10.8. For six different proofs see Steele's article [33]. Our proof of Theorem 10.8 was modeled after Hammersley's [16] proof of the upper bound and Erdös-Szekeres's proof of the lower bound.

If c is small then $TR(k,c)$ is substantially smaller than $R(k,c)$. This indicates that the Transitive Ramsey's Theorem is weaker than Ramsey's Theorem. We speculate that, in some cases, by using the transitive Ramsey's Theorem, a proof of termination may also provide a bound on run time. This is also possible for Ramsey's Theorem; however, using the transitive Ramsey's Theorem, one may be able to obtain a much better bound.

11 Solving Subcases of the Termination Problem

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

Def 11.1 Let $M_1^{()}, M_2^{()}, \ldots$ be a standard list of oracle Turing Machine. They do not take inputs. (We interpret the oracle as the user-supplied inputs.) If $A \subseteq \mathsf{N}$ and $s \in \mathsf{N}$ then $M_{i,s}^A \uparrow$ means that if you run M_i^A it will not halt within s steps. Let $M_1^{()}, M_2^{()}, \ldots$ be a standard list of oracle Turing Machine.

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

Def 11.2

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

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

Program 5

```
(Comment: X is (x[1],...,x[n])
X = (input(INT),input(INT),...,input(INT))
While x[1]>0 and x[2]>0 and ... x[n]>0
 control = input(1,2,3,...,m)
  if control==1
   X = (g_11(X,input(INT),...,input(INT))
  else
  if control==2
   X = (g_21(X,input(INT),...,input(INT))
  else
    .
    .
    .
  else
  if control==m
   X =(g_m1(X,input(INT),...,input(INT))
```

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 are known:

Theorem 11.3

1. *$TERM$ is Π^1_1-complete*

2. *If X is Π^1_1-complete then, for all Y in the arithmetic hierarchy, $Y \leq_m X$.*

3. *For all Y in the arithmetic hierarchy $Y \leq_m TERM$. (This follows from (1) and (2).)*

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 N^n to N. Let $m \in \mathsf{N}$. A *$(F(n), m)$-program* is a program of the form of Program 5 where, for all $1 \leq i \leq m$

$$(g_{i1}, \ldots, g_{in}) \in FUN(n)^n.$$

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 for the **While** loop condition $Mx \geq b$ for some matrix M and vector b; however, such programs can easily be transformed into programs of our form.

1. Tiwari [34] has shown that the following problem is decidable: Given matrices A, B and vector c, all over the rationals, is the following program in $TERM$ (interpreting the variables over the reals).

Program 6

```
x = input(RAT)
while (Bx > b)
        x = Ax + c
```

Since this is a program over the reals it does not fit into our framework.

2. Braverman [5] has shown that the following problem is decidable: Given matrices A, B_1, B_2 and vectors b, b_1, c, all over the rationals, is the following program in $TERM$ (interpreting the variables over the rationals.)

Program 7

```
x = input(RAT)
while (B1x > b) AND (B2x ≥ b1)
        x = Ax + c
```

In the homogeneous case, where $b_1 = b_2 = c = 0$ the result holds where the domain is the integers. Hence our problem is decidable if $m = 1$ and $FUN(n)$ consists of linear functions with no constant term.

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

Program 8

```
x=INPUT(INT)
while (Bx ≥ b)
        if x[i] ≥ 0
                then x = A0x
        else
                x = A1x
```

This can be formulated as an undecidability result in our framework.

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
     $$\mathrm{f(x[1],\ldots,x[n])} = \min\{\mathrm{x[i_1]} + \mathrm{c_1}, \mathrm{x[i_2]} + \mathrm{c_2}, \ldots, \mathrm{x[i_k]} + \mathrm{c_k}\}$$
     where $1 \le \mathrm{i_1} < \cdots < \mathrm{i_k}$ and $\mathrm{c_1}, \ldots, \mathrm{c_k} \in \mathsf{Z}$.

- 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 Notice that Program 3 falls into this category.

  f(x[1],. . . ,x[n])= x[i]+c

  where $1 \leq i \leq n$ and c$\in$ Z. Program 3 falls into this category.

The results above contrast 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 Σ_1^1 complete.

12 Appendix: Termination of Program 3, Using Theorem 6.4

This section is due to Amir Ben-Amram.

The case control=1 is represented by the matrix

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

and the other case by

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

To show that Thm 6.4 holds, we form a representation of the closure set closA, b, that is the set of all finite products of A and B's. The representation (obtained according to (Ben-Amram 2008)) is:

$$\mathcal{E} = CD^x, \qquad \text{where } C \in \{A, B, AB, BA\} \text{ and } D = \begin{pmatrix} -1 & \infty \\ \infty & -1 \end{pmatrix}.$$

To apply the theorem, it suffices that the representation *contains* the closure set. This is easily proved by induction. We multiply each of the four "patterns" in $\mathcal{E}$ *on the left* by each of the matrices. We use the following identities: $A^2 = DA = AD = AB$, $B^2 = D$, $DB = BD$.

1. $A(AD^x) = AD^{x+1}$
2. $B(AD^x) = (BA)D^x$
3. $A(BD^x) = (AB)D^x$
4. $B(BD^x) = BD^{x+1}$
5. $A(ABD^x) = ADBD^x = ABD^{x+1}$

6. $B(ABD^x) = BAD^{x+1}$
7. $A(BAD^x) = A^3D^x = AD^{x+2}$
8. $B(BAD^x) = DAD^x = AD^{x+1}$

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. $AD^x = \begin{pmatrix} -1-x & -x \\ \infty & \infty \end{pmatrix}$
2. $(BD^x)^2 = B^2D^{2x} = D^{2x+1}$
3. $ABD^x = ADD^x = AD^{x+1}$
4. $BAD^x = \begin{pmatrix} \infty & \infty \\ -3 & -2 \end{pmatrix} D^x = \begin{pmatrix} \infty & \infty \\ -3 & -2-x \end{pmatrix}$.

13 Acknowledgments

I would like to thank Daniel Apon, Amir Ben-Amram, Peter Cholak, Byron Cook, Denis Hirschfeldt, Jon Katz, Andreas Podelski, Andrey Rybalchenko, and Richard Shore for helpful discussions.

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.

[5] M. Braverman, editor. *Termination of integer linear programs*. Springer, 2006.

[6] 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).

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

[8] 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/.

[9] 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/.

[10] 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/.

[11] 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/.

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

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

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

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

[16] 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.

[17] 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.

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

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

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

[21] 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.

[22] 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.

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

[24] J.-Y. Moyen. SCT and the idempotence condition, 2011. http://www-lipn.univ-paris13.fr/~moyen/publications.html.

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

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

[27] 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/.

[28] 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/.

[29] S. Radziszowski. Small Ramsey numbers. *The electronic journal of combinatorics*, 2000. Located at www.combinatorics.org.

[30] 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.

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

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

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

[34] A. Tiwari, editor. *Termination of linear programs*, volume 3115 of *Lecture Notes in Computer Science*. Springer, 2004.