On the Termination of Integer Loops

Amir M. Ben-Amram[1], Samir Genaim[2], and Abu Naser Masud[3]

[1] School of Computer Science, The Tel-Aviv Academic College, Israel
[2] DSIC, Complutense University of Madrid (UCM), Spain
[3] DLSIIS, Technical University of Madrid (UPM), Spain

Abstract. In this paper we study the decidability of termination of several variants of simple integer loops, without branching in the loop body and with affine constraints as the loop guard (and possibly a precondition). We show that termination of such loops is undecidable in some cases, in particular, when the body of the loop is expressed by a set of linear inequalities where the coefficients are from $\mathbb{Z} \cup \{r\}$ with r an arbitrary irrational; or when the loop is a sequence of instructions, that compute either linear expressions or the step function. The undecidability result is proven by a reduction from counter programs, whose termination is known to be undecidable. For the common case of integer constraints loops with rational coefficients only we have not succeeded in proving decidability nor undecidability of termination, however, this attempt led to the result that a Petri net can be simulated with such a loop, which implies some interesting lower bounds. For example, termination for a given input is at least EXPSPACE-hard.

1 Introduction

Termination analysis has received a considerable attention and nowadays several powerful tools for the automatic termination analysis of different programming languages and computational models exist [15,12,1,25]. Two important aspects of termination analysis tools are their scalability and ability to handle a large class of programs, which are directly related to the theoretical limits, regarding complexity and completeness, of the underlying techniques. Since termination of general programs is undecidable, every attempt at solving it in practice will have at its core certain restricted problems, or classes of programs, that the algorithm designer targets. To understand the theoretical limits of an approach, we are looking for the decidability and complexity properties of these restricted problems. Note that understanding the boundaries set by inherent undecidability or intractability of problems yields more profound information than evaluating the performance of one particular algorithm.

Much of the recent development in termination analysis has benefited from techniques that deal with one simple loop at a time, where a simple loop is specified by (optionally) some initial conditions, a loop guard, and a "loop body" of a very restricted form. Very often, the state of the program during the loop is represented by a finite set of scalar variables (this simplification may be the result of an abstraction, such as size abstraction of structured data [25,11]).

V. Kuncak and A. Rybalchenko (Eds.): VMCAI 2012, LNCS 7148, pp. 72–87, 2012.

Regarding the representation of the loop body, the most natural one is, perhaps, a block of straight-line code, namely a sequence of assignment statements, as in the following example:

$$\textit{while } X > 0 \textit{ do } \{\, X := X + Y;\, Y := Y - 1;\, \} \tag{1}$$

To define a restricted problem for theoretical study, one just has to state the types of loop conditions and assignments that are admitted.

By symbolically evaluating the sequence of assignments, a straight-line loop body may be put into the simple form of a simultaneous deterministic update, namely loops of the form

$$\textit{while } C \textit{ do } \langle x_1, \ldots, x_n \rangle := f(\langle x_1, \ldots, x_n \rangle)$$

where f is a function of some restricted class. For function classes that are sufficiently simple to analyze, one can hope that termination of such loops would be decidable; in fact, the main motivation to this paper has been the remarkable results by Tiwari [26] and Braverman [10] on the termination of *linear loops*, a kind of loops where the update function f is linear. The loop conditions in these works are conjunctions of linear inequalities. Specifically, Tiwari proved that the termination problem is decidable for loops of the following form:

$$\textit{while } (B\boldsymbol{x} > \boldsymbol{b}) \textit{ do } \boldsymbol{x} := A\boldsymbol{x} + \boldsymbol{c} \tag{2}$$

where the arithmetic is done over the reals; thus the variable vector $\boldsymbol{x}$ has values in $\mathbb{R}^n$, and the constant matrices in the loop are $B \in \mathbb{R}^{m \times n}$, $A \in \mathbb{R}^{n \times n}$, $\boldsymbol{b} \in \mathbb{R}^m$ and $\boldsymbol{c} \in \mathbb{R}^n$.

Consequently, Braverman proved decidability of termination of loops of the following form:

$$\textit{while } (B_s\boldsymbol{x} > \boldsymbol{b}_s) \wedge (B_w\boldsymbol{x} \geq \boldsymbol{b}_w) \textit{ do } \boldsymbol{x} := A\boldsymbol{x} + \boldsymbol{c} \tag{3}$$

where the constant matrices and vectors are *rational*, and the variables are of either real or rational type; moreover, in the homogeneous case ($\boldsymbol{b}_s, \boldsymbol{b}_w, \boldsymbol{c} = 0$) he proved decidability when the variables range over $\mathbb{Z}$. This is a significant and non-trivial addition, since algorithmic methods that work for the reals often fail to extend to the integers (a notorious example is finding the roots of polynomials; while decidable over the reals, over the integers, it is the undecidable *Hilbert* 10^{th} *problem*[1]).

Going back to program analysis, we note that it is typical in this field to assume that some degree of approximation is necessary in order to express the effect of the loop body by linear arithmetics alone. Hence, rather than loops with a linear update as above, one defines the representation of a loop body to be a set of *constraints* (again, usually linear). The general form of such a loop is

$$\textit{while } (B\boldsymbol{x} \geq \boldsymbol{b}) \textit{ do } A \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{x'} \end{pmatrix} \leq \boldsymbol{c} \tag{4}$$

[1] Over the rationals, the problem is still open, according to [18].

where the loop body is interpreted as expressing a relation between the new values $\boldsymbol{x}'$ and the previous values $\boldsymbol{x}$. Thus, in general, this representation is a non-deterministic kind of program and may super-approximate the semantics of the source program analyzed. But this is a form which lends itself naturally to analysis methods based on linear programming techniques, and there has been a series of publications on proving termination of such loops [24,19,22] — all of which rely on the generation of *linear ranking functions*. For example, the termination analysis tools *Terminator* [12], COSTA [1], and Julia [25], are based on proving termination of such loops by means of a linear ranking function.

It is known that the linear-ranking approach cannot completely resolve the problem [22,10], since there are terminating programs having no such ranking function, e.g., the loop (1) above. Moreover, the linear-programming based approaches are not sensitive to the assumption that the data are integers. Thus, the problem of decidability of termination for linear constraint loops (4) stays open, in its different variants. We feel that the most intriguing problem is:

> Is the termination of a single linear constraints loop decidable, when the coefficients are rational (or integer) numbers and the variables range over the integers?

The problem may be considered for a given initial state, for any initial state, or for a (linearly) constrained initial state.

Our contribution. In this research, we focused on hardness proofs. Our basic tool is a new simulation of counter programs (also known as counter machines) by a simple integer loop. The termination of counter programs is a well-known undecidable problem. While we have not been able to fully answer the major problem above, this technique led to some interesting results which improve our understanding of the simple-loop termination problem. We next summarize our main results. All concern integer variables.

1. We prove undecidability of termination, either for all inputs or a given input, for simple loops which iterate a straight-line sequence of simple assignment instructions. The right-hand sides are integer linear expressions except for one instruction type, which computes the step function

$$f(x) = \begin{cases} 0 & x \leq 0 \\ 1 & x > 0 \end{cases}$$

   At first sight it may seem like the inclusion of such an instruction is tantamount to including a branch on zero, which would immediately allow for implementing a counter program. This is not the case, because the result of the function is put into a variable which can only be combined with other variables in a very limited way. We complement this result by pointing out other natural instructions that can be used to simulate the step function. This include integer division by a constant (with truncation towards zero) and truncated subtraction.

2. Building upon the previous result, we prove undecidability of termination, either for all inputs or for a given input, of linear constraint loops where *one irrational number may appear* (more precisely, the coefficients are from $\mathbb{Z} \cup \{r\}$ for an arbitrary irrational number r).
3. Finally, we observe that while linear constraints with rational coefficients seem to be insufficient for simulating *all* counter programs, it is possible to simulate a subclass, namely Petri nets, leading to the conclusion that termination for a given input is at least EXPSPACE-hard.

We would like to highlight the relation of our results to the discussion at the end of [10]. Braverman notes that constraint loops are non-deterministic and asks:

> How much non-determinism can be introduced in a linear loop with no initial conditions before termination becomes undecidable?

It is interesting that our reduction to constraint loops, when using the irrational coefficient, produces constraints that are *deterministic*. The role of the constraints is not to create non-determinism; it is to express complex relationships among variables. We may also point out that some limited forms of linear constraint loops (that are very non-deterministic since they are weaker constraints) have a *decidable* termination problem (see Section 6). Braverman also discusses the difficulty of deciding termination for a given input, a problem that he left open. Our results apply to this variant, providing a partial answer to this open problem.

The rest of this paper is organized as follows. Section 2 presents some preliminaries; Section 3 study the termination of straight-line while loops with a "built-in" function that represents the step function; Section 4 attempts to apply the technique of Section 3 to the case of integer constraints loops, and discusses extensions of integer constraints loops for which termination is undecidable; Section 5 describes how a Petri net can be simulated with linear constraint loops; Section 6 discusses some related work; and Section 7 concludes.

2 Preliminaries

In this section we define the syntax of integer piecewise linear while loops, integer linear constraints loops, and counter programs.

2.1 Integer Piecewise Linear Loops

An integer piecewise linear loop (*IPL* loop for short) with integer variables $X_1, \ldots, X_n$ is a while loop of the form

$$\mathit{while}\ b_1 \wedge \cdots \wedge b_m\ \mathit{do}\ \{c_1; \ldots; c_n\}$$

where each condition b_i is a linear inequality $a_0 + a_1 * X_1 + \cdots + a_n * X_n \geq 0$ with $a_i \in \mathbb{Z}$, and each c_i is one of the following instructions

$$X_i := X_j + X_k \mid X_i := a * X_j \mid X_i := a \mid X_i = \mathit{isPositive}(X_j)$$

such that $a \in \mathbb{Z}$ and

$$isPositive(X) = \begin{cases} 0 & X \leq 0 \\ 1 & X > 0 \end{cases}$$

We consider *isPositive* to be a primitive, but in the next section we will consider alternatives. The semantics of an *IPL* loop is the obvious: starting from initial values for the variables $X_1, \ldots, X_n$ (the input), the instructions $c_1, \ldots, c_n$ are executed sequentially as far as the condition $b_1 \wedge \cdots \wedge b_n$ holds. We say that the loop terminates for a given input if the condition $b_1 \wedge \cdots \wedge b_n$ eventually evaluates to *false*. For simplicity, sometime we use a composite expression, e.g, $X_1 := 2 * X_2 + 3 * X_3 + 1$, which should be taken to be a syntactic sugar for a series of assignments, possibly using temporary variables. We will also make use of a "macro" $isZero(X)$ which should be understood as representing the expression $1 - isPositive(X) - isPositive(-X)$.

2.2 Integer Linear Constraints Loops

An integer linear constraints loop (*ILC* loop for short) over n variables $\boldsymbol{x} = \langle X_1, \ldots, X_n \rangle$ has the form

$$while\ (B\boldsymbol{x} \geq \boldsymbol{b})\ do\ A \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{x'} \end{pmatrix} \leq \boldsymbol{c}$$

where for some $m, p > 0$, $B \in \mathbb{R}^{m \times n}$, $A \in \mathbb{R}^{p \times 2n}$, $\boldsymbol{b} \in \mathbb{R}^m$ and $\boldsymbol{c} \in \mathbb{R}^p$. The case we are most interested in is that in which the constant matrices and vectors are composed of rational numbers; this is equivalent to assuming that they are all integers (just multiply by a common denominator).

Semantically, a state of such a loop is an n-tuple $\langle x_1, \ldots, x_n \rangle$ of integers, and a transition to a new state $\boldsymbol{x'} = \langle x'_1, \ldots, x'_n \rangle$ is possible if $\boldsymbol{x}, \boldsymbol{x'}$ satisfy all the constraints in the loop guard and the loop body. We say that the loop terminates for given initial state if all possible executions from that state are finite, and that it universally terminates if it terminates for every initial state. We say that the loop is *deterministic* if there is at most one successor state to any state.

2.3 Counter Programs

A (deterministic) counter program P_C with n counters $X_1, \cdots, X_n$ is a list of labeled instructions $1{:}I_1, \ldots, m{:}I_m, m{+}1{:}stop$ where each instruction I_k is one of the following:

$$incr(X_j) \mid decr(X_j) \mid if\ X_i > 0\ then\ k_1\ else\ k_2$$

with $1 \leq k_1, k_2 \leq m{+}1$ and $1 \leq j \leq n$. A state is of the form $(i, \langle a_1, \ldots, a_n \rangle)$ which indicates that Instruction i is to be executed next, and the current values of the counters are $X_1 = a_1, \ldots, X_n = a_n$. In a valid state, $1 \leq i \leq m+1$ and all $a_i \in \mathbb{N}$ (it will sometimes be useful to also consider invalid states, and assume that they cause a halt). Any state in which $i = m + 1$ is a halting state. For any other valid state $(i, \langle a_1, \ldots, a_n \rangle)$, the successor state is defined as follows.

- If I_i is $decr(X_j)$ (resp. $incr(X_j)$), then X_j is increased (resp. decreased) by 1 and the execution moves to label $i+1$.
- If I_i is "*if* $X_j > 0$ *then* k_1 *else* k_2" then the execution moves to label k_1 if X_j is positive, and to k_2 if it is 0. The values of the counters do not change.

For simplicity, we assume that a counter with value 0 is never decremented, this can be guaranteed by adding a conditional statement before each $decr(X_j)$. The following are known facts about the halting problem for counter programs.

Theorem 1 ([21]). *The halting problem for counter programs with $n \geq 2$ counters and the initial state $(1, \langle 0, \ldots, 0 \rangle)$ is undecidable.*

The *universal halting problem* is the problem of deciding whether a given program halts for any initial state.

Theorem 2 ([7]). *The universal halting problem for counter programs with $n \geq 2$ counters is undecidable.*

3 Termination of *IPL* Loops

In this section, we investigate the decidability of the following problems: given an *IPL* loop P

1. Does P terminate for a given input?
2. Does P terminate for all inputs?

We show that both problems are undecidable by reduction from the halting problem for counter programs. To see where the challenge in this reduction lies, note that the loops we iterate a fixed block of straight-line code, while a counter program has a program counter that determines the next instruction to execute. While one can easily keep the value of the PC in a variable (which is what we do), it is not obvious how to make the computation depend on this variable, and how to simulate branching.

3.1 The Reduction

Given a counter program $P_C \equiv 1{:}I_1, \ldots, m{:}I_m, m{+}1{:}\mathit{stop}$ with counters $X_1, \ldots, X_n$, we generate a corresponding *IPL* loop $\mathcal{T}(P_C)$ as follows:

while $(PC \geq 1 \wedge PC \leq m \wedge X_1 \geq 0 \wedge \cdots \wedge X_n \geq 0)$ **do** {
  $\mathcal{T}(1{:}I_1)$
  $\vdots$
  $\mathcal{T}(m{:}I_m)$
  $PC := N_1 + \cdots + N_m;$
}

where $\mathcal{T}(k{:}I_k)$ is defined as follows

– If $I_k \equiv incr(X_j)$, then $\mathcal{T}(k{:}I_k)$ is

  $A_k := isZero(PC - k)$;
  $X_j := X_j + A_k$;
  $N_k := (k+1) * A_k$;

– If $I_k \equiv decr(X_j)$, then $\mathcal{T}(k{:}I_k)$ is

  $A_k := isZero(PC - k)$;
  $X_j := X_j - A_k$;
  $N_k := (k+1) * A_k$;

– If $I_k \equiv$ *if* $X_j > 0$ *then* k_1 *else* k_2, then $\mathcal{T}(k{:}I_k)$ is

  $A_k := isZero(PC - k)$;
  $F_k := isPositive(X_j)$;
  $T_k := isPositive(A_k + F_k - 1)$;
  $N_k := T_k * (k_1 - k_2) + A_k * k_2$;

In Section 3.2 we prove the following:

Lemma 1. *A counter program P_C with $n \geq 2$ counters terminates for the initial state $(i, \langle a_1, \ldots, a_n \rangle)$ if and only if $\mathcal{T}(P_C)$ terminates for the initial input $PC = i \wedge X_1 = a_1 \wedge \cdots \wedge X_n = a_n$.*

Lemma 1, together with theorems 1 and 2, imply.

Theorem 3. *The halting problem and universal halting problem for IPL loops are undecidable.*

3.2 Proof of Correctness

Let us first state, informally, the main ideas behind the reduction, and then formally prove Lemma 1 which in turn implies Theorem 3.

1. Variable PC represents the program counter, i.e., the label of the instruction to be executed next.
2. Variables $A_1, \ldots, A_m$ are flags: when $PC = i$, then $A_k = 1$ if $k = i$, and $A_k = 0$ if $k \neq i$. Thus, an operation $X_j := X_j + A_k$ (resp. $X_j := X_j - A_k$) will have effect only when $A_k = 1$, and otherwise it is a no-op. This is a way of simulating only one instruction in every iteration.
3. Variables $N_1, \ldots, N_m$ are used to compute the value of PC for the next iteration. The idea is that when $PC = k$, N_k is set to the label of the next instruction; while N_i, for $i \neq k$, is set to 0. Thus, the new PC can be obtained by summing these variables.

Note that point (3) guarantees that $PC := N_1 + \cdots + N_m$ correctly computes the label of the next instruction. Thus, the while loop simulates the execution of the counter program. Now we move to the formal proof.

Lemma 2. *Let $PC = i$; then for all k, A_k is set to 1 when $k = i$, and to 0 when $k \neq i$.*

Proof. Immediate from the semantics of *isZero* and the code that sets A_k.

Lemma 3. *Let $PC = i$, then (1) for $k \neq i$, it holds that $N_k = 0$; and (2) for $k = i$, it holds that $N_k = k+1$ if I_k is $decr(X_j)$ or $incr(X_j)$, and $N_k = k_1$ (resp. $N_k = k_2$) if I_k is "if $X_j > 0$ then k_1 else k_2" and $X_j > 0$ (resp. $X_j = 0$).*

Proof. We consider the following two cases

1. Assume $k \neq i$, then (a) for $incr(X_j)$ and $decr(X_j)$ it is obvious that $N_k = 0$ since by Lemma 2 we have $A_k = 0$; and (b) for "*if* $X_j > 0$ *then* k_1 *else* k_2", since A_k equals 0 by Lemma 2, then also $T_k = 0$ (regardless of the value of F_k), and thus $N_k = 0 * (k_1 - k_2) + 0 * k_2 = 0$;
2. Assume $k = i$, then (a) for $incr(X_j)$ and $decr(X_j)$ it is obvious that $N_k = k+1$ since by Lemma 2 we have $A_k = 1$; and (b) for "*if* $X_j > 0$ *then* k_1 *else* k_2", by Lemma 2 we have $A_k = 1$, and by definition of *isPositive* we have $F_k = 0$ when $X_j = 0$ and $F_k = 1$ when $X_j > 0$. Thus

X_j	F_k	T_k	N_k
> 0	1	1	$1 * (k_1 - k_2) + 1 * k_2 = k_1$
$= 0$	0	0	$0 * (k_1 - k_2) + 1 * k_2 = k_2$

In order to prove Lemma 1, it is enough to show that $\mathcal{T}(P_C)$ simulates the corresponding counter program P_C.

Lemma 4. *Let P_C be a counter program, $\mathcal{T}(P_C)$ its corresponding IPL loop, $C \equiv (\ell, \langle a_1, \ldots, a_n \rangle)$ a configuration for P_C, and S a state of $\mathcal{T}(P_C)$ where $PC = \ell, X_1 = a_1, \ldots, X_n = a_n$. Then C is a halting configuration of P_C if and only if S terminates $\mathcal{T}(P_C)$; while if C has a successor state $(\ell', \langle a'_1, \ldots, a'_n \rangle)$ in P_C, then the loop body of $\mathcal{T}(P_C)$ is enabled at S and its execution leads to a state in which $PC = \ell', X_1 = a'_1, \ldots, X_n = a'_n$.*

Proof. For invalid initial states both the counter programs and the corresponding while loop terminate immediately. For valid states the proof follows from Lemmata 2 and 3.

3.3 Examples of Piecewise-Linear Operations

The *isPositive* operation can be easily simulated by other natural instructions, yielding different instruction sets that suffice for undecidability.

Example 1 (Integer division). Consider an instruction that divides an integer by an integer constant and truncates the result towards zero (also if it is negative). Using this kind of division, we have

$$isPositive(X) = X - \frac{2 * X - 1}{2}$$

and thus, termination is undecidable for loops with linear assignments and integer division of this kind.

Example 2 (truncated subtraction). Another common piecewise-linear function is *truncated subtraction*, such that $x \dot{-} y$ is the same as $x - y$ if it is positive, and otherwise 0. This operation allows for implementing *isPositive* thus:

$$isPositive(X) = 1 \dot{-} (1 \dot{-} X)$$

4 Reduction to *ILC* Loops

In this section we turn to Integer Linear Constraint loops. We attempt to apply the reduction described in Section 3, and explain where and why it fails. So we do not obtain undecidability for *ILC* loops, but we show that if there is one irrational number that we are allowed to use in the constraints (any irrational will do) the reduction can be completed and undecidability of termination proved.

In Section 5 we describe another way of handling the failure of the reduction with rational coefficients only: reducing from a weaker model, and thereby proving a lower bound which is weaker than undecidability (but still non-trivial).

Observe that the loop constructed in Section 3 uses non-linear expressions only for setting the flags A_k, F_k and T_k, the rest is clearly linear. Assuming that we can encode these flags with integer linear constraints, then adapting the rest of the reduction to *ILC* loops is straightforward: it can be done by rewriting $\mathcal{T}(P_C)$ to avoid multiple updates of a variable (that is, to *single static assignment form*) and then representing each assignment as an equation instead. Thus, in what follows we concentrate on how to represent those flags using integer linear constraints.

4.1 Encoding T_k with Integer Linear Constraints

In Section 3, we defined T_k as $isPositive(A_k + F_k - 1)$. Since $0 \leq A_k + F_k \leq 2$, it is easy to verify that this is equivalent to imposing the constraint $F_k + A_k - 1 \leq 2 \cdot T_k \leq F_k + A_k$.

4.2 Encoding A_k with Integer Linear Constraints

The role of the flag A_k is to indicate if PC is equal to k. Expressing this relation by linear constraints is possible thanks to the finite range of PC, as shown by the following lemma.

Lemma 5. *Let P_1 and P_2 be the following polyhedra*

$$P_1 = \bigwedge_i (A_i \geq 0) \wedge (A_1 + ... + A_m = 1)$$
$$P_2 = (PC = 1 \cdot A_1 + 2 \cdot A_2 + \cdots + m \cdot A_m)$$

Then $P_1 \wedge P_2 \wedge (PC = k) \rightarrow (A_k = 1)$ and $P_1 \wedge P_2 \wedge PC \neq k \rightarrow (A_k = 0)$.

Proof. It is easy to see that the only integer points in P_1 are such that for a single k, $A_k = 1$, while for all $j \neq k$, $A_j = 0$. Then, P_2 forces PC to equal k.

4.3 Encoding F_k with Integer Linear Constraints

Now we discuss the difficulty of encoding the flag F_k using linear constraints. The following lemma states that such encoding is not possible when using rational coefficients.

Lemma 6. *Given non-negative integer variables X and F, it is not possible to define a system of integer linear constraints Ψ (with rational coefficients) over X, F, and possibly other integer variables, such that $\Psi \wedge (X = 0) \rightarrow (F = 0)$ and $\Psi \wedge (X > 0) \rightarrow (F = 1)$.*

Proof. The proof relies on a theorem in [20] which states that the following piecewise linear function

$$f(x) = \begin{cases} 0 & x = 0 \\ 1 & x > 0, \end{cases}$$

where x is a non-negative *real* variable, cannot be defined as a minimization mixed integer programming (*MIP* for short) problem with rational coefficients only. More precisely, it is not possible to define $f(x)$ as

$$f(x) = \text{minimize } g \text{ w.r.t. } \Psi$$

where Ψ is a system of linear constraints with rational coefficients over x and other integer and real variables, and g is a linear function over $vars(\Psi)$. Now suppose that Lemma 6 is false, i.e., there exists Ψ such that $\Psi \wedge (X = 0) \rightarrow (F = 0)$ and $\Psi \wedge (X > 0) \rightarrow (F = 1)$, then the following *MIP* problem

$$f(x) = \text{minimize } F \text{ w.r.t. } \Psi \wedge (x \leq X)$$

defines the function $f(x)$, which contradicts [20].

4.4 An Undecidable Extension of *ILC* Loops

There are certain extensions of the *ILC* model that allow our reduction to be carried out. Basically, the extension should allow for encoding the flag F_k. The extension which we find most interesting allows the use of a single, arbitrary irrational number r (thus, we do not require the specific value of r to represent any particular information). Thus, the coefficients are now over $\mathbb{Z} \cup \{r\}$. The variables still hold integers.

Lemma 7. *Let r be an arbitrary positive irrational number, and let*

$$\Psi_1 = (0 \leq F_k \leq 1) \wedge (F_k \leq X)$$
$$\Psi_2 = (rX \leq B) \wedge (rY \leq A) \wedge (-Y \leq X) \wedge (A + B \leq F_k)$$

then $(\Psi_1 \wedge \Psi_2 \wedge X = 0) \rightarrow F_k = 0$ and $(\Psi_1 \wedge \Psi_2 \wedge X > 0) \rightarrow F_k = 1$.

Proof. The constraints Ψ_1 force F_k to be 0 when X is 0, and when X is positive F_k can be either 0 or 1. The role of Ψ_2 is to eliminate the non-determinism for the case $X > 0$, namely, for $X > 0$ it forces F_k to be 1. The property that makes Ψ_2 work is that for a given *non-integer* number d, the condition $-A \leq d \leq B$ implies $A + B \geq 1$, whereas for $d = 0$ the sum may be zero. The role of the irrational coefficient is to translate any integer value X, except 0, to a non-integer number: $d = rX$ (similarly also for Y and rY). The variable Y is introduced to avoid using another irrational coefficient $-r$.

Example 3. Let us consider $r = \sqrt{2}$ in lemma 7. When $X = 0$, Ψ_1 forces F_k to be 0, and it is easy to verify that Ψ_2 is satisfiable for $X = Y = A = B = F_k = 0$. Now, for the positive case, let for example $X = 5$, then Ψ_1 limits F_k to the values 0 or 1, and Ψ_2 implies $(\sqrt{2} \cdot 5 \leq B) \wedge (-\sqrt{2} \cdot 5 \leq A)$ since $Y \geq -5$. The minimum values that A and B can take are respectively -7 and 8, thus it is not possible to choose A and B such that $A + B \leq 0$. This eliminates $F_k = 0$ as a solution. However, for these minimum values we have $A + B = 1$ and thus $A + B \leq F_k$ satisfiable for $F_k = 1$.

Theorem 4. *The termination of ILC loops where the coefficients are from* $\mathbb{Z} \cup \{r\}$*, for a single arbitrary irrational constant* r*, is undecidable.*

We have mentioned, above, Meyer's result that *MIP* problems with rational coefficients cannot represent the step function over reals. Interestingly, he also shows that it *is* possible using an irrational constant, in a manner similar to our Lemma 7. Our technique construction differs in that we do not make use of minimization or maximization to define the function.

5 Simulation of Petri Nets

Let us consider a counter machine as defined in Section 2, but with a *weak* conditional statement "*if* $X_j < a$ *then* k_1 *else* k_2" (where a is a positive integer) which is interpreted as: if X_j is smaller than a then the execution *must* continue at label k_1, otherwise it *may* continue to label k_1 or label k_2. This computational model is equivalent to a Petri net. From considerations as those presented in Section 4, we arrived at the conclusion that the weak conditional, and therefore Petri nets, *can* be simulated by an *ILC* loop.

A (place/transition) Petri net [23] is composed of a set of counters $X_1, \ldots, X_n$ (known as *places*) and a set of transitions $t_1, \ldots, t_m$. A transition is essentially a command to increment or decrement some places. This may be represented formally by associating with transition t its set of decremented places ${}^\bullet t$ and its set of incremented places $t^\bullet$. A transition is said to be enabled if all its decremented places are non-zero, and it can then be *fired*, causing the decrements and increments associated with it to take place. Starting from an initial marking (values for the places), the state of the net evaluates by repeatedly firing one of the enabled transitions.

Lemma 8. *Given a Petri net P with initial marking M, a simulating ILC loop with an initial condition Ψ_M can be constructed in polynomial time. In particular, the termination of the loop from an initial state in Ψ_M is equivalent to the termination of P starting from M.*

How this is done: The *ILC* loop will have variables $X_1, \ldots, X_n$ that represent the counters in a straight-forward way, and flags $A_1, \ldots, A_m$ that represent the choice of the next transition much as we did for counter programs (except that there is no *PC* variable). For each $1 \leq i \leq n$ we create the following constraints in the body of the loop:

$$P_1 = \bigwedge_k (A'_k \geq 0) \wedge (A'_1 + ... + A'_m = 1)$$

$$\Psi_i = \bigwedge_i (X_i \geq \sum_{k:i \in {}^\bullet t_k} A'_k)$$

$$\Phi_i = (X'_i = X_i - \sum_{k:i \in {}^\bullet t_k} A'_k + \sum_{k:i \in t_k{}^\bullet} A'_k)$$

The loop guard is $X_1 \geq 0 \wedge \cdots \wedge X_n \geq 0$. The initial state Ψ_M simply forces each X_i to have the value as stated by the initial marking M. Note that the initial values of A_i are not important since they are not used (we only use A'_k). As before, the constraint P_1 ensures that one and only one of the A'_k will equal 1 at every iteration. The constraints Ψ_i ensure that A'_k may receive the value 1 only if transition k is enabled in the state. The constraints Φ_i (the update) clearly simulate the chosen transition.

The importance of this result is that complexity results for Petri net are lower bounds on the complexity of the corresponding problems in the context of *ILC* loops, and in particular, from a known results about the termination problem [14,17], we obtain the following.

Theorem 5. *The halting problem for ILC loops, for a given input, is at least EXPSPACE-hard.*

Note that the reduction does not provide useful information on universal termination of *ILC* loops, since for Petri net it is PTIME-decidable [13].

6 Related Work

Termination of integer loops has received considerable attention recently, both from theoretical (e.g., decidability, complexity), and practical (e.g, developing tools) perspectives. Research has considered: straight-line while loops, and loops in a constraint setting possibly with multiple-paths.

For straight-line while loops, the most remarkable results are those of Tiwari [26] and Braverman [10]. Tiwari proved that the problem is decidable for linear deterministic updates when the domain of the variables is $\mathbb{R}$. Braverman proved that this holds also for $\mathbb{Q}$, and for the homogeneous case it holds for $\mathbb{Z}$

(see discussion in Section 1). Both have considered universal termination, the termination for a given input left open.

Decidability and complexity of termination of single and multiple-path integer linear constraints loops has been intensively studied for different classes of constraints. Lee et al. [16] proved that termination of a multiple-path *ILC* loop, when the constraints are restricted to size-change constraints (i.e., constraints of the form $X_i > X'_j$ or $X_i \geq X'_j$ over $\mathbb{N}$), is PSPACE-complete [16]. Later, Lee and Ben-Amram [6] identified sub-classes of such loops for which the termination can be decided in polynomial time. Ben-Amram [4] showed how to extend and adapt some theory from the domain of size-change constraints to general monotonicity constraints (i.e., constraints of the form $X_i > Y$, $X_i \geq Y$, where Y can be primed or unprimed variable), he proved that termination for such loops is PSPACE-complete. It is important to note that his results hold for any well-founded domain, not necessarily $\mathbb{N}$. In [5], Ben-Amram considered loops with monotonicity constraints over $\mathbb{Z}$, and prove that the termination problem is PSPACE-complete. Recently, Bozzelli and Pinchinat [8] proved that it is still PSPACE-complete also for gap-constraints, i.e., constraints of the form $X - Y \geq c$ where $c \in \mathbb{N}$. Ben-Amram [3] proved that when extending size-change constraints with integer constants, i.e., allowing difference constraints of the form $X_i - X'_j \geq c$ where $c \in \mathbb{Z}$, the termination problem become undecidable. However for a subclass in which each source (i.e., unprimed) variable might be used only once (in each path) the problem is PSPACE-complete.

All the above work concerns multiple-path loops. Petri nets and various extensions, such as Reset and Transfers nets, can also be seen as multiple-path constraint loops. The termination of Petri net and several extensions is known to be decidable [13,14,17].

Back to single-path loops, a topic that received much attention is the synthesis of ranking functions for such loops, as a means of proving termination. Sohn and Van Gelder [24] proposed a method for the synthesis of linear ranking functions for *ILC* loops over $\mathbb{N}$. Later, their method was extended by Mesnard and Serebrenik [19] to $\mathbb{Z}$ and to multiple-path loops. Both rely on the duality theorem of linear programming. Podelski and Rybalchenko [22] also proposed a method for synthesizing linear ranking function for *ILC* loops. Their method is based on Farkas' lemma. It is important to note that [19,22] are complete when the variables range over $\mathbb{R}$ or $\mathbb{Q}$, but not $\mathbb{Z}$. Recently, Bagnara et al. [2] proved that [19,22] are actually equivalent, in the sense that they compute the same set of ranking functions, and that the method of Podelski and Rybalchenko has better worst-case complexity. Bradley et al. [9] presented an algorithm for computing linear ranking functions for straight-line integer while loops with integer division.

Piecewise affine functions have been long used to describe the step of a discrete time dynamical system. Blondel el al. [7] considered systems of the form $x(t+1) = f(x(t))$ where f is a piecewise affine function over $\mathbb{R}^n$ (defined by rational coefficients). They show that some problems are undecidable for $n \geq 2$,

in particular, whether all trajectories go through 0 (the moartality problem). This can be seen as termination of the loop `while` $x \neq 0$ `do x := f(x)`.

7 Conclusion

Motivated by the increasing interest in the termination of integer loops, in this research, we have studied the hardness of terminations proofs for several variants of such loops. In particular, we have considered straight-line while loops, and integer linear constraints loops. The later are very common in the context of program analysis.

For straight-line while loops, we proved that if the underlying instructions set allows the implementation of a simple piecewise linear function, namely the step function, then the termination problem is undecidable. For integer linear constraints loops, we have showed that allowing the constraints to include a single arbitrary irrational number makes the termination problem undecidable. For the case of integer constraints loops with rational coefficients only, which is very common in program analysis, we could simulate a Petri net. This result provide interesting lower bound on the complexity of the termination, and other related problems, of *ILC* loops.

We have recently obtained additional results using techniques similar to those described in this paper. Specifically, we have shown an EXPSPACE lower bound, as in Section 5, that holds for *ILC* loops with a deterministic update. We have also shown undecidability for a while loop having the body of the following form

`if` ($x > 0$) `then` (one deterministic update) `else` (another update)

and the guard as in *IPL* loops, where the updates are linear (and do not involve the step function).

We hope that our results shed some light on the termination problem of simple integer loops and perhaps will inspire further progress on the open problems.

Acknowledgments. We thank Pierre Ganty for discussions on Petri nets. Amir Ben-Amram's work was done while visiting DIKU, the department of Computer Science at the University of Copenhagen. This work was funded in part by the Information & Communication Technologies program of the EC, Future and Emerging Technologies (FET), under the ICT-231620 *HATS* project, by the Spanish Ministry of Science and Innovation (MICINN) under the TIN-2008-05624 *DOVES* project, the UCM-BSCH-GR35/10-A-910502 *GPD* Research Group and by the Madrid Regional Government under the S2009TIC-1465 *PROMETIDOS-CM* project.

References

1. Albert, E., Arenas, P., Genaim, S., Puebla, G., Zanardini, D.: COSTA: Design and Implementation of a Cost and Termination Analyzer for Java Bytecode. In: de Boer, F.S., Bonsangue, M.M., Graf, S., de Roever, W.-P. (eds.) FMCO 2007. LNCS, vol. 5382, pp. 113–132. Springer, Heidelberg (2008)

2. Bagnara, R., Mesnard, F., Pescetti, A., Zaffanella, E.: The automatic synthesis of linear ranking functions: The complete unabridged version. Quaderno 498, Dipartimento di Matematica, Università di Parma, Italy, Published as arXiv:cs.PL/1004.0944 (2010)
3. Ben-Amram, A.M.: Size-change termination with difference constraints. ACM Trans. Program. Lang. Syst. 30(3) (2008)
4. Ben-Amram, A.M.: Size-change termination, monotonicity constraints and ranking functions. Logical Methods in Computer Science 6(3) (2010)
5. Ben-Amram, A.M.: Monotonicity constraints for termination in the integer domain. CoRR, abs/1105.6317 (2011)
6. Ben-Amram, A.M., Lee, C.S.: Program termination analysis in polynomial time. ACM Trans. Program. Lang. Syst. 29(1) (2007)
7. Blondel, V.D., Bournez, O., Koiran, P., Papadimitriou, C.H., Tsitsiklis, J.N.: Deciding stability and mortality of piecewise affine dynamical systems. Theor. Comput. Sci. 255(1-2), 687–696 (2001)
8. Bozzelli, L., Pinchinat, S.: Verification of Gap-Order Constraint Abstractions of Counter Systems. In: Kuncak, V., Rybalchenko, A. (eds.) VMCAI 2012. LNCS, vol. 7148, pp. 88–103. Springer, Heidelberg (2012)
9. Bradley, A.R., Manna, Z., Sipma, H.B.: Termination Analysis of Integer Linear Loops. In: Abadi, M., de Alfaro, L. (eds.) CONCUR 2005. LNCS, vol. 3653, pp. 488–502. Springer, Heidelberg (2005)
10. Braverman, M.: Termination of Integer Linear Programs. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 372–385. Springer, Heidelberg (2006)
11. Bruynooghe, M., Codish, M., Gallagher, J.P., Genaim, S., Vanhoof, W.: Termination analysis of logic programs through combination of type-based norms. ACM Trans. Program. Lang. Syst. 29(2) (2007)
12. Cook, B., Podelski, A., Rybalchenko, A.: Termination proofs for systems code. In: Schwartzbach, M.I., Ball, T. (eds.) Proceedings of the ACM SIGPLAN 2006 Conference on Programming Language Design and Implementation (PLDI), Ottawa, Canada, pp. 415–426. ACM (2006); Terminator
13. Dufourd, C., Jančar, P., Schnoebelen, P.: Boundedness of Reset P/T Nets. In: Wiedermann, J., Van Emde Boas, P., Nielsen, M. (eds.) ICALP 1999. LNCS, vol. 1644, pp. 301–310. Springer, Heidelberg (1999)
14. Esparza, J.: Decidability and Complexity of Petri Net Problems—an Introduction. In: Reisig, W., Rozenberg, G. (eds.) APN 1998. LNCS, vol. 1491, pp. 374–428. Springer, Heidelberg (1998)
15. Giesl, J., Thiemann, R., Schneider-Kamp, P., Falke, S.: Automated Termination Proofs with AProVE. In: van Oostrom, V. (ed.) RTA 2004. LNCS, vol. 3091, pp. 210–220. Springer, Heidelberg (2004)
16. Lee, C.S., Jones, N.D., Ben-Amram, A.M.: The size-change principle for program termination. In: POPL, pp. 81–92 (2001)
17. Lipton, R.J.: The reachability problem requires exponential space. Technical Report 63, Yale University (1976), http://www.cs.yale.edu/publications/techreports/tr63.pdf
18. Matiyasevich, Y.: Hilbert's tenth problem: What was done and what is to be done. In: Denef, J., Lipshitz, L., Pheidas, T., Van Geel, J. (eds.) Hilbert's Tenth Problem: Relations with Arithmetic and Algebraic Geometry. AMS (2000)
19. Mesnard, F., Serebrenik, A.: Recurrence with affine level mappings is p-time decidable for clp(r). TPLP 8(1), 111–119 (2008)

20. Meyer, R.R.: Integer and mixed-integer programming models: General properties. Journal of Optimization Theory and Applications 16, 191–206 (1975), doi:10.1007/BF01262932
21. Minsky, M.L.: Computation: finite and infinite machines. Prentice-Hall, Inc., Upper Saddle River (1967)
22. Podelski, A., Rybalchenko, A.: A Complete Method for the Synthesis of Linear Ranking Functions. In: Steffen, B., Levi, G. (eds.) VMCAI 2004. LNCS, vol. 2937, pp. 239–251. Springer, Heidelberg (2004)
23. Reisig, W.: Petri Nets: An Introduction. EATCS Monographs on Theoretical Computer Science. Springer, Berlin (1985)
24. Sohn, K., Van Gelder, A.: Termination detection in logic programs using argument sizes. In: PODS, pp. 216–226. ACM Press (1991)
25. Spoto, F., Mesnard, F., Payet, É.: A termination analyzer for java bytecode based on path-length. ACM Trans. Program. Lang. Syst. 32(3) (2010)
26. Tiwari, A.: Termination of Linear Programs. In: Alur, R., Peled, D.A. (eds.) CAV 2004. LNCS, vol. 3114, pp. 70–82. Springer, Heidelberg (2004)