Termination Proofs for Linear Simple Loops*

Hong Yi Chen[1], Shaked Flur[2], and Supratik Mukhopadhyay[1]

[1] Department of Computer Science
Louisiana State University
Baton Rouge, LA 70803
hchen11@lsu.edu, supratik@csc.lsu.edu
[2] Department of Computer Science
The Technion
Haifa 32000, Israel
fshaked@cs.technion.ac.il

Abstract. Analysis of termination and other liveness properties of an imperative program can be reduced to termination proof synthesis for simple loops, *i.e.*, loops with only variable updates in the loop body. Among simple loops, the subset of *Linear Simple Loops* (LSLs) is particular interesting because it is common in practice and expressive in theory. Existing techniques can successfully synthesize a linear ranking function for an LSL if there exists one. However, when a terminating LSL does not have a linear ranking function, these techniques fail. In this paper we describe an automatic method that generates proofs of universal termination for LSLs based on the synthesis of disjunctive ranking relations. The method repeatedly finds linear ranking functions on parts of the state space and checks whether the transitive closure of the transition relation is included in the union of the ranking relations. Our method extends the work of Podelski and Rybalchenko [27]. We have implemented a prototype of the method and have shown experimental evidence of the effectiveness of our method.

1 Introduction

Termination proof synthesis for simple loops, *i.e.*, loops with only variable updates in the loop body, are the building blocks of the liveness analysis of large complex systems [16, 29, 17, 10, 23, 26, 25, 28, 22, 24]. In particular, we consider a subclass of simple loops which contain only linear updates with the flexibility of handling nondeterminism. We call them *Linear Simple Loops* (LSLs). LSLs are interesting because most loops in practice are indeed linear; more importantly, with its capability to handle nondeterminism LSLs are expressive enough to serve as a foundational model for other simple loops.

* This research is partially supported by NSF under the grant 0965024. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

A. Miné and D. Schmidt (Eds.): SAS 2012, LNCS 7460, pp. 422–438, 2012.

It is well known that termination of simple loops with linear guards and linear assignments (they form the deterministic subclass of LSL) over rationals or reals is decidable [34]. The termination problem for "homogeneous cases" over integers, of the deterministic LSL subclass is also decidable [9]. Ben-Amram *et al.* recently proved termination of LSLs is undecidable when the the coefficients are from $\mathbb{Z} \cup \{r\}$ with r being an arbitrary irrational number [2]. However, when we reduce the analysis of a complex system to that of an LSL, knowing whether or not the LSL terminates is not enough, we often need to obtain a termination proof, such as a ranking function or a ranking relation, that the overall analysis can build upon [16, 29, 17, 10, 23, 26, 25, 28]. When it comes to finding termination proofs for LSLs, Podelski and Rybalchenko's technique [27] can generate a linear ranking function if there exists one. This method is based on Farkas's lemma [33] that provides a technique to derive hidden constraints from a system of linear inequalities. The method is complete when LSLs range over rationals or reals. However, if a terminating LSL has only non-linear ranking functions, this method will return failure. In this paper, we extend the method of Podelski and Rybalchenko, and solve cases for which only non-linear ranking functions exist.

Our approach is closely related to the previous work on termination proof synthesis based on disjunctive ranking relations. The traditional method for proving program termination, proposed by Turing [35], relies on proving $R \subseteq \tau(f)$, where R is the program's transition relation, $\tau(f)$ is a ranking relation given by a ranking function f. The difficulty with Turing's method is that a single ranking function is usually hard to find. With LSLs in particular, non-linear ranking functions are often difficult to synthesize. To address this problem Podelski and Rybalchenko proposed [28] proving $R^+ \subseteq \tau(f_1) \cup \cdots \cup \tau(f_n)$, where R^+ is the transitive closure of R, and $\tau(f_1) \cup \cdots \cup \tau(f_n)$ is a finite union of ranking relations. Many recent approaches for proving termination for general programs are based on disjunctive ranking relations [3, 4, 1, 18, 17, 29, 26, 25]. In this paper, instead of trying to synthesize a single non-linear ranking function, we generate disjunctive linear ranking functions and check the validity of $R^+ \subseteq \tau(f_1) \cup \cdots \cup \tau(f_n)$.

To be able to apply the disjunctive ranking relation proof rule [28], we need the technique of binary reachability check (BRC). That is, given a disjunctive ranking relation T, we need to prove or disprove the inclusion $R^+ \subseteq T$. We use the technique developed by the TERMINATOR team [17] to check this inclusion. Their approach is to syntactically transform the program so that binary reachability check is reduced to unary reachability check, which is a well studied task and can be carried out on any temporal safety checker. Moreover, if the validity of $R^+ \subseteq T$ is not satisfied, the construction of the transformed program will enable the safety checker to generate an error path that violates the inclusion. In our problem setting, if the input to BRC is an LSL, the error path will induce a new LSL which is an unrolling of the original LSL. Thus we can repeatedly check for binary reachability and expand the current disjunctive ranking relation.

In this paper we provide a method for automatically generating disjunctive ranking relations as proofs of universal termination for LSLs. Roughly speaking, the idea is to repeatedly partition the state space based on trace segments [19, 31, 21] such that one of the subspace is guaranteed to have a linear ranking function. The partitioning will generate a series of linear ranking functions f_i's such that $R \subseteq \bigcup \tau(f_i)$. However this does not suffice as a termination proof since the inclusion should refer to R^+ and not to R. For a termination proof we leverage BRC; if $R^+ \subseteq \bigcup \tau(f_i)$ is satisfied, it returns success; if not, BRC provides a new LSL, and we look for the next series of ranking functions f_j's. We then again check for the inclusion $R^+ \subseteq \bigcup \tau(f_i) \cup \bigcup \tau(f_j)$. This process continues until we successfully find an over-approximation of R^+. The question that remains is how to effectively find the sub-space for which a linear ranking function exists. To answer this question, we resort to the simple fact that when variables range over $\mathbb{Z}$ and updates are deterministic, two constraints $x \geq b$ and $x > x'$, where b is a number, x' represents the value of x after one transition, guarantees x to be a ranking function. Similarly, whenever we have a constraint of the form $\varphi \geq b$, we partition the state space by constraint $\varphi > \textsc{Shift}(\varphi)$ (function Shift is formally defined in Section 4.2) and its negation $\varphi \leq \textsc{Shift}(\varphi)$. When the variables range over $\mathbb{Q}$ or $\mathbb{R}$, or updates are nondeterministic, more complicated partition needs to be performed, as is described in Section 3.2 and 4.2.

Lastly, we provide experimental results showing that our method outperforms both linear ranking function synthesis [27] and polyranking method [7] on a suite of LSL examples provided in [11].

Related Work. Rather than looking for disjunctive ranking relations as the termination proof, Cousot [20] shows how non-linear ranking functions can be synthesized over nonlinear loops based on the S-procedure for semi-definite programming. Colón and Sipma's work on linear loops with multiple paths and assertional transition relations achieve to synthesize linear ranking functions via polyhedral manipulation in [13, 14]. Bradley *et al.* show how to synthesize lexicographic linear ranking functions with supporting linear invariants over loops with linear assertional transition relations in [6]. Another type of termination proof is polyranking functions raised by Bradley *et al.* A polyranking function needs not always decrease but decreases eventually. It is a generalization of the regular polynomial ranking function. In [8], the authors show a method for finding bounded expressions that are eventually negative over loops with parallel transitions. In [7], the authors demonstrate a method for synthesize lexicographic linear polyranking functions with supporting linear invariants over linear loops.

Other related works include proving conditional termination, which aims to find a set of initial states, usually an underapproximation of it, that guarantees termination. Cook *et al.* in [15] proposed an approach that first finds potential ranking functions then solves for the sub-space that guarantees the potential ranking function to be a true ranking function. Bozga *et al.* represent the set of non terminating states in terms of greatest fixpoint and then utilize quantifier elimination to deduce the exact set and consequently the dual set (*i.e.* the terminating states).

2 Preliminaries

2.1 Loop Model and Semantics

Through out this paper, all variables range over domain $\mathbb{Z}$, $\mathbb{Q}$, or $\mathbb{R}$. The following definition provides the syntax of LSLs.

Definition 1 (Linear Simple Loops). *A Linear Simple Loop over program variables $X^0 = (x_1, x_2, \ldots, x_m)$ and its n copies $X^1, X^2, \ldots, X^n$ $(m, n \geq 1)$ is a tuple $L = \langle \textsc{Cond}, \textsc{Update}, i, j \rangle$ where*

- Cond *is a set of linear constraints of the form $a_i X^i \bowtie b$ $(\bowtie \in \{<, \leq, =, \geq, >\})$.*
- Update *is a set of linear constraints of the form $a_0 X^0 + \cdots + a_n X^n \bowtie b$ $(\bowtie \in \{<, \leq, =, \geq, >\})$.*
- *i and j are integers and $0 \leq i < j \leq n$.*
- *a_k and b are coefficients that range over $\mathbb{Z}$ or $\mathbb{Q}$.*

We sometime refer to Cond and Update as loop conditions and loop updates respectively. Intuitively, L describes unrolling of a loop (and maybe some extra constraints) with a back edge from j to i.

The formal semantics of LSL is defined as follows. Let $L = \langle \textsc{Cond}, \textsc{Update}, i, j \rangle$ be an LSL over variables X^0 and its n copies $X^1, X^2, \ldots, X^n$. An $(n+1)$-trace of L is a tuple $(s_0, s_1, \ldots, s_n)$ such that all the constraints in Cond and Update are satisfied simultaneously when assigning s_0 to X^0, s_1 to X^1, $\ldots$, s_n to X^n. We denote by $R_{n+1}(L)$ the set of all $(n+1)$-traces of L and $R(L)$ the relation L describes.

$$R(L) = \{(s_i, s_j) \mid (s_0, s_1, \ldots, s_n) \in R_{n+1}(L)\}$$

The most simple LSL $L = \langle \textsc{Cond}, \textsc{Update}, 0, 1 \rangle$ involves only X^0 and X^1. Without explicitly stating i and j, by default we assume $i = 0,\ j = 1$. An $L = \langle \textsc{Cond}, \textsc{Update}, 0, 1 \rangle$ describes a transition relation from a state that is before a transition (given by X^0) to the corresponding state that is after the transition (given by X^1). For example, the following while loop

$$\begin{aligned} &\mathit{while}\ (x > 0) \\ &\quad x := x - 1; \end{aligned}$$

can be rewritten as the following LSL

$$L_1 \triangleq \langle \{x^0 > 0\}, \{x^1 = x^0 - 1\}, 0, 1 \rangle$$

An $L = \langle \textsc{Cond}, \textsc{Update}, 0, 1 \rangle$ can also have more than two copies of variables. For example,

$$L_2 \triangleq \langle \{x^0 > 0\}, \{x^1 = x^0 - 1,\ x^1 > 0,\ x^2 = x^1 - 1,\ x^2 \leq 0\}, 0, 2 \rangle$$

X^1 and X^0 represent the values of the variable X after and before a transition, respectively. The constraint $x^2 \leq 0$ restrains the input space of L_2 to $x^0 \in \{2\}$.

Note that $x^0 = 1$ is not in our input space, since there does not exist x^1 and x^2 satisfying L_2. In our approach, we add such constraints over future transition states such that the restrained LSL is guaranteed to have a linear ranking function.

An LSL $L = \langle \textsc{Cond}, \textsc{Update}, 0, k\rangle$ describes a transition relation between X^0 and X^k. For example, in

$$L_3 \triangleq \langle \{x^0 > 0, \}, \{x^1 = x^0 - 1,\ x^1 > 0,\ x^2 = x^1 - 1\}, 0, 2\rangle$$

the transition pairs described by L_3 include: $(5, 3), (4, 2), (3, 1), \ldots$. We often use $L = \langle \textsc{Cond}, \textsc{Update}, 0, k\rangle$ when we are looking at the k-th unrolling of some $L' = \langle \textsc{Cond}', \textsc{Update}', 0, 1\rangle$.

To further generalize our loop model, we provide the ability not only to look ahead, but also to look back.

$$L_4 \triangleq \langle \{x^1 > 0\}, \{x^0 > 0,\ x^1 = x^0 - 1,\ x^2 = x^1 - 1,\ x^2 \leq 0\}, 1, 2\rangle$$

Despite having the same constraints as in L_2, the input space of $L_4(1, 2)$ is $x^1 \in \{1\}$.

Note that LSLs allow nondeterminism. To be specific, we can have linear expressions on both sides of an update statement, and inequalities instead of equal relation. This gives us more flexibility to model nondeterministic inputs or non-linear operations. For example, we can have an LSL such as

$$L_5 \triangleq \langle \{x^0 > 0\}, \{x^0 + x^1 \leq 1\}, 0, 1\rangle$$

that cannot be expressed in any conventional programming language.

2.2 Disjunctive Ranking Relations

Definition 2 (Well-Ordered Sets). *A set D is well-ordered with respect to a relation $<$ if,*

1. *$<$ is a strict total ordered and,*
2. *There is no infinite sequence $d_0, d_1, d_2, \ldots$ of elements in D such that $d_{i+1} < d_i$ for every $i \in \mathbb{N}$.*

Definition 3 (Ranking Functions). *Given a transition relation $R \subseteq S \times S$, a function $r : S \to D$ is a ranking function, if D is a well-ordered set and for every $(s_1, s_2) \in R$ we have $r(s_2) < r(s_1)$, where $<$ is the well order associated with D.*

A Ranking Function is called linear (Linear Ranking Function) if r is linear.

Definition 4 (Ranking Relations). *Given a ranking function $r : S \to D$ we define the corresponding ranking relation by*

$$\tau(r) = \{(s_1, s_2) \mid r(s_2) < r(s_1)\}$$

where $<$ is the well order associated with D.

Definition 5 (Disjunctive Ranking Relations). *A disjunctive ranking relation T is a finite union of ranking relations. That is,*

$$T = T_1 \cup \cdots \cup T_n$$

where T_i is a ranking relation for $1 \leq i \leq n$, $n \in \mathbb{N}$

The relation between disjunctive ranking relations and termination has been established in [28] using Ramsey's theorem [30]. Let P be a program, R be the corresponding transition relation induced by P, R^+ be the non-reflexive transitive closure of R, then P is terminating if and only if

$$R^+ \subseteq T$$

for some disjunctive ranking relation T.

2.3 Binary Reachability Check

Given an LSL L and a disjunctive ranking relation T, the goal of binary reachability check is to verify whether $R^+(L) \subseteq T$. If yes, the procedure returns "true". Otherwise, the procedure returns an error path which induces a new LSL L' such that L' is an unrolling of L and $R(L') \nsubseteq T$. The input and output of procedure BRC is described as follows (see [17] for more details):

input
LSL L, disjunctive ranking relation T
output
if $(R^+(L) \subseteq T)$ return "true"
else return LSL L' such that L' is an unrolling of L and $R(L') \nsubseteq T$

2.4 Simple Linear Ranking Function Synthesis

The following theorem is proved in [27].

Theorem. *An LSL given by the system $(A^0 A^1)\binom{x^0}{x^1} \leq b$ (i.e. $i = 0, j = 1$) is terminating if there exist nonnegative vectors λ_1, λ_2 over rationals such that the following system is satisfiable:*

$$\lambda_1 A^1 = 0 \tag{1}$$

$$(\lambda_1 - \lambda_2)(A^0) = 0 \tag{2}$$

$$\lambda_2(A^0 + A^1) = 0 \tag{3}$$

$$\lambda_2 b < 0 \tag{4}$$

More over, the LSL has a linear ranking function of the form

$$\rho(X^0) = \begin{cases} rX^0 & \text{if exists } X^1 \text{ such that } (A^0 A^1)\binom{x^0}{x^1} \leq b \\ \delta_0 - \delta & \text{otherwise} \end{cases}$$

where $r \triangleq \lambda_2 A^1$, $\delta_0 \triangleq -\lambda_1 b$, and $\delta \triangleq -\lambda_2 b$.

We will extend this method in Section 4 so that it works for the general form of LSLs.

3 Example

We first demonstrate our technique with a simple deterministic LSL over the integers. Then we will extend our technique for nondeterministic updates and rational / real variables.

3.1 Deterministic Updates over Integer Domain

Consider the while loop in Figure 1. It has only 3 simple assignments, but it is not obvious whether it is terminating. It is easy to see that the traces of z are composed of two alternating numbers, one negative the other non-negative, and that the negative number has a higher value. The variable y always gets assigned to value of z from the previous state. Hence it behaves like z, except being one step behind. The variable x increments itself with y. Therefore x will alternatively increase (or stay unchanged) and decrease. Moreover the decrease is larger than the increase, hence x will eventually become negative and the loop will terminate.

```
int x, y, z;
while (x ≥ 0)
    x := x + y;
    y := z;
    z := -z - 1;
```

Fig. 1. Example

Let us first convert the while loop above to an LSL.

$$L = \langle \{x^0 \geq 0\}, \{x^1 = x^0 + y^0,\ y^1 = z^0,\ z^1 = -z^0 - 1\}, 0, 1 \rangle$$

If we apply the method of Section 2.4 to L, it will return failure since L does not have a linear ranking function. As mentioned earlier, we want to construct multiple linear ranking functions, each of them over a restrained input space. We do this by adding constraints to L such that the new LSL is guaranteed to have a linear ranking function. From $\textsc{Cond}_L$ we see that we already have the linear expression x^0 that is bounded, *i.e.*, $x^0 \geq 0$. If we add to L a constraint $x^0 > x^1$, then we know x^0 can serve as a ranking function for the restrained LSL because x^0 has a lower bound and is strictly decreasing, which is a sufficient condition for x^0 to become a ranking function over the integer domain.

We break L into two LSLs $L_{1.1}$ and $L_{1.2}$ such that $L_{1.1}$ is obtained by combining L with constraint $x^0 > x^1$, and $L_{1.2}$ is obtained by combining L with the negation of the constraint, namely $x^0 \leq x^1$.

$$L_{1.1} = \langle \textsc{Cond}_L, \textsc{Update}_L \cup \{x^0 > x^1\}, 0, 1 \rangle \qquad \text{(trivial case)}$$
$$L_{1.2} = \langle \textsc{Cond}_L, \textsc{Update}_L \cup \{x^0 \leq x^1\}, 0, 1 \rangle \qquad \text{(synthesis case)}$$

We call $L_{1.1}$ the *trivial case* since we immediately obtain a linear ranking function from it.

$$\rho_1(X^0) = \begin{cases} x^0 & \text{if } \exists X^1 \text{ such that } X^0, X^1 \text{ satisfies } L_{1.1} \\ -1 & \text{otherwise} \end{cases}$$

We call $L_{1.2}$ the *synthesis case* since it needs further examination. We call $\textsc{Cond}_L$ the *Seed* for partitioning L.

From this point onwards, we only need to take care of $L_{1.2}$. First we check whether $L_{1.2}$ has a linear ranking function already. In this particular case we find out that this is not true. Next, we would like to repeat the earlier process on $L_{1.2}$, *i.e.*, adding constraints to $L_{1.2}$ such that a linear ranking function must exist. Since $L_{1.2}$ already includes $x^0 \leq x^1$, using $x^0 \geq 0 \wedge x^0 > x^1$ again will no longer make sense. However observe that the new constraint in $L_{1.2}$ gives a new linear expression that is bounded below, *i.e.*, $x^1 \geq x^0$. This constraint will become our new *Seed*, and we can use it to partition $L_{1.2}$. This time we partition with the constraint $(x^1 - x^0) > (x^2 - x^1)$ and its negation.

At this point a new issue arises, x^2 is introduced to denote the value of x after one transition from x^1. However from $L_{1.2}$ alone, there is no such information about x^2. To remedy this situation, we first need to unroll L so that the unrolled transition involves x^2. We do this by making a copy of all the loop constraints in L, then changing X^1 to X^2, X^0 to X^1 (the process is formally described by $\textsc{Unroll}(L)$ in Section 4). We get L_2 as follows.

$$\begin{aligned} L' =& \textsc{Unroll}_2(L) \\ =& \langle \{x^0 \geq 0\}, \{x^1 = x^0 + y^0,\ y^1 = z^0,\ z^1 = -z^0 - 1, \\ & \quad x^1 \geq 0,\ x^2 = x^1 + y^1,\ y^2 = z^1,\ z^2 = -z^1 - 1\}, 0, 1\rangle \\ L_2 =& \langle \textsc{Cond}_{L'}, \textsc{Update}_{L'} \cup \mathit{Seed}, 0, 1\rangle \\ =& \langle \{x^0 \geq 0\}, \{x^1 = x^0 + y^0,\ y^1 = z^0,\ z^1 = -z^0 - 1, \\ & \quad x^1 \geq 0,\ x^2 = x^1 + y^1,\ y^2 = z^1,\ z^2 = -z^1 - 1\} \cup \{x^1 \geq x^0\}, 0, 1\rangle \end{aligned}$$

Now we can partition L_2 using the constraint mentioned above.

$$\begin{aligned} L_{2.1} =& \langle \textsc{Cond}_{L_2}, \textsc{Update}_{L_2} \cup \{(x^1 - x^0) > (x^2 - x^1)\}, 0, 1\rangle \quad \text{(trivial case)} \\ L_{2.2} =& \langle \textsc{Cond}_{L_2}, \textsc{Update}_{L_2} \cup \{(x^1 - x^0) \leq (x^2 - x^1)\}, 0, 1\rangle \quad \text{(synthesis case)} \end{aligned}$$

$L_{2.1}$ is again the trivial case, where a linear ranking function is guaranteed

$$\rho_2(X^0) = \begin{cases} x^1 - x^0 = y^0 & \text{if } \exists X^1, X^2 \text{ such that } X^0, X^1, X^2 \text{ satisfies } L_{2.1} \\ 0 & \text{otherwise} \end{cases}$$

Now we check whether the synthesis case has a linear ranking function. Notice that this time we can not use the method described in Section 2.4 any more, since now the synthesis case LSL involves X^2. In Section 4, we describe a general ranking function synthesis method which can handle this general form of LSLs.

If we feed $L_{2.2}$ to the method in Section 4, we get the following linear ranking function.

$$\rho_3(X^0) = \begin{cases} 2x^0 + z^0 & \text{if } \exists X^1, X^2 \text{ such that } X^0, X^1, X^2 \text{ satisfies } L_{2.2} \\ -1 & \text{otherwise} \end{cases}$$

As shown in Figure 2, up to this point we have divided L to three LSLs, $L_{1.1}$, $L_{2.1}$, and $L_{2.2}$. Each of these three has a linear ranking function. Let $T = \tau(\rho_1) \cup \tau(\rho_2) \cup \tau(\rho_3)$. Theorem 2 in Section 4 shows us that $R(L) \subseteq T$. That is, any two consecutive states form a pair that belongs to T.

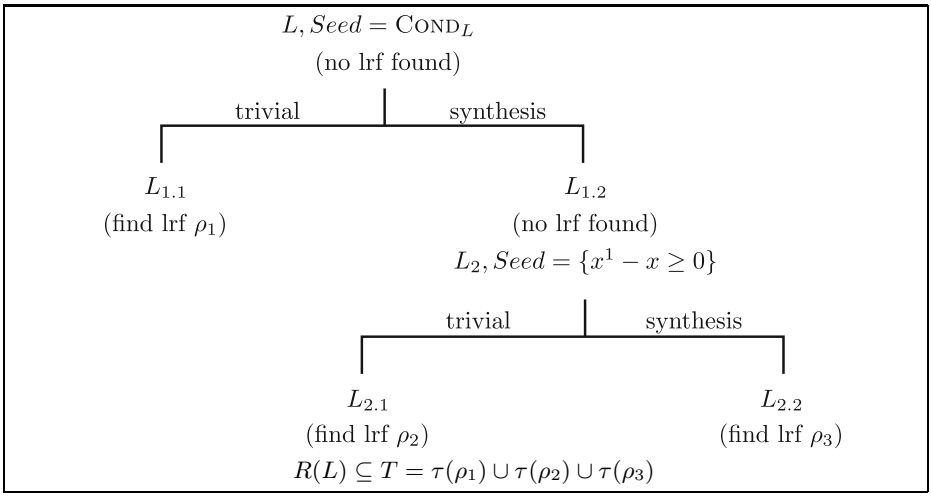

Fig. 2. Execution of $L \triangleq \langle \{x^0 \geq 0\}, \{x^1 = x^0 + y^0, y^1 = z^0, z^1 = -z^0 - 1\}, 0, 1 \rangle$

Recall that our goal is to find a T such that $R^+(L) \subseteq T$. We first check whether the T we found already satisfies $R^+(L) \subseteq T$. As it turns out for this particular case, it is not. BRC gives an error path that executes L twice. Therefore we get a new LSL L'' that unrolls L twice and L'' describes a relation from X^0 to X^2.

$$L'' = \text{UNROLL}_2(L) \text{ with } i_{L''} = 0, j_{L''} = 2$$

Note L'' has the same set of constraints as L', but has different backedge. We feed L'' to the method described in Section 4.1. It shows that L'' has a linear ranking function already.

$$\rho_4(X^0) = \begin{cases} x^0 + y^0 & \text{if } \exists X^1, X^2 \text{ such that } X^0, X^1, X^2 \text{ satisfies } L' \\ -1 & \text{otherwise} \end{cases}$$

Again we update T by $T = T \cup \tau(\rho_4)$ and this time the test $R^+(L) \subseteq T$ succeeds, *i.e.*, we have successfully found a disjunctive ranking relation T for the original LSL L.

3.2 Variables over $\mathbb{Q}$ or $\mathbb{R}$ and Nondeterministic Updates

Notice that when variables range over $\mathbb{Q}$ or $\mathbb{R}$, the two constraints $\varphi \geq b$ and $\varphi > \textsc{Shift}(\varphi)$ can no longer guarantee φ to be a linear ranking function. One way to remedy that is to pick a small positive value c and partition the state space by $\varphi - \textsc{Shift}(\varphi) > c$ and its negation $\varphi - \textsc{Shift}(\varphi) \leq c$. Similar to the integer example, the former constraint will generate the trivial case, and the latter constraint will generate the synthesis case.

Another way is to still partition with $\varphi > \textsc{Shift}(\varphi)$ and its negation $\varphi \leq \textsc{Shift}(\varphi)$. However since the former can no longer generate a trivial case, we need to continue the partition process on the trivial case as well.

Nondeterministic updates are also an issue. If we look at ranking function ρ_2 above, the expression y^0 originates from the expression $x^1 - x^0$. We cannot use $x^1 - x^0$ directly because the ranking functions need to be expressed in terms of X^0. With deterministic updates, we can get rid of x^1 by substituting it with $x^0 + y^0$. With nondeterministic updates, we may not be able to simplify the expression in this manner. Therefore we need to apply Theorem 1 in Section 4.1 to generate ranking functions on X^0 only, and when we fail to find one, we need to partition the trivial case further. In our algorithm shown in Figure 4, this is the approach we take in all situations.

4 Algorithm for Synthesizing Disjunctive Ranking Relations

4.1 Extended Linear Ranking Function Synthesis

Let $\boldsymbol{A}$ denote the row vector $(A^0 \dots A^i \dots A^j \dots A^n)$, $\boldsymbol{A}_i$ denote the i-th element A^i, $\boldsymbol{A}_{-i}$ denote the row vector with all but the i-th element $(A^0 \dots A^{i-1} A^{i+1} \dots A^j \dots A^n)$. Similarly we define column vectors $\boldsymbol{X}$, $\boldsymbol{X}_i$, and $\boldsymbol{X}_{-i}$. Then we prove the following theorem.

Theorem 1. *An LSL $L = \langle \textsc{Cond}, \textsc{Update}, i, j \rangle$ given by the system $\boldsymbol{AX} \leq b$ is terminating if there exist non-negative vectors λ_1, λ_2 over rationals such that the following system is satisfied:*

$$\lambda_1 \boldsymbol{A}_{-i} = 0 \tag{1}$$
$$\lambda_2 \boldsymbol{A}_{-i,-j} = 0 \tag{2}$$
$$(\lambda_1 - \lambda_2)\boldsymbol{A}_i = 0 \tag{3}$$
$$\lambda_2(\boldsymbol{A}_i + \boldsymbol{A}_j) = 0 \tag{4}$$
$$\lambda_2 b < 0 \tag{5}$$

More over, the LSL has a linear ranking function of the form

$$\rho(X^i) = \begin{cases} rX^i & \text{if exists } X^j \text{ such that } (X^i, X^j) \in R(L) \\ \delta_0 - \delta & \text{otherwise} \end{cases}$$

where $r \triangleq \lambda_2 \boldsymbol{A}_j$, $\delta_0 \triangleq -\lambda_1 b$, and $\delta \triangleq -\lambda_2 b$.

Note that Theorem 1 cannot be replaced by using the Theorem in Section 2.4 because Theorem 1 guarantees to generate ranking functions expressed in X^i only, while the original theorem does not. Just as in [27], the converse of the above theorem is true for rationals and reals. Since this paper does not focus on the application of the converse theorem, we do not elaborate it here.

4.2 Formal Description

To help describing the algorithm, we need to define a few notations here. We start by defining SHIFT which is the process of transforming constraint from a certain copy of X to a higher copy. It does so by incrementing the superscript of each X^i. For example, $\textsc{Shift}(x^0 - x^1 < 1) = x^1 - x^2 < 1$.

Definition 6 (Shift). *Given a linear combination $\psi : a_0X^0+a_1X^1+\cdots+a_nX^n$, a linear constraint $\varphi : \psi \leq b$, a set of linear constraints C, where $b \in \mathbb{Z}$, we define*

$$\begin{aligned}
\textsc{Shift}(\psi) &\triangleq a_0X^1 + a_1X^2 + \cdots + a_nX^{n+1} \\
\textsc{Shift}_1(\varphi) &\triangleq \textsc{Shift}(\psi) \leq b \\
\textsc{Shift}_{k+1}(\varphi) &\triangleq \textsc{Shift}_k(\textsc{Shift}(\psi) \leq b) \\
\textsc{Shift}_k(C) &\triangleq \{\textsc{Shift}_k(\varphi) \mid \varphi \in C\}
\end{aligned}$$

Next we define function UNROLL. This function produces an LSL with the same traces as the original but with more copies of X. It does so by adding SHIFT of the constraints to itself. Note that function UNROLL is used in the partitioning process (see L' in Section 3.1). The BRC procedure also unrolls an LSL (see L'' in Section 3.1). The only difference between the two unrolling is that BRC changes the value of j to the number of iterations in the error path.

Definition 7 (Unroll). *Given a set of linear constraints C, an LSL $L = \langle \textsc{Cond}, \textsc{Update}, i, j\rangle$, we define*

$$\begin{aligned}
\textsc{Unroll}_{1,d}(C) &\triangleq C \\
\textsc{Unroll}_{k+1,d}(C) &\triangleq C \cup \textsc{Shift}_d(\textsc{Unroll}_{k,d}(C)) \\
\textsc{Unroll}_k(L) &\triangleq \langle \textsc{Cond}, \textsc{Unroll}_{k,j-i}(\textsc{Cond} \cup \textsc{Update}), i, j\rangle
\end{aligned}$$

Lastly we define function DIFF. This function creates new constraints that we use to partition the original LSL. It does so by taking constraint, shifting it and then binding the constraint and its shift with $>$ or $\leq$. For instance, for constraint $\varphi : x^0 \leq 0$ we have $\textsc{Diff}_{1,>}(\varphi) = x^1 > x^0$ and $\textsc{Diff}_{1,\leq}(\varphi) = x^1 \leq x^0$.

Definition 8 (Diff). *Given a linear constraint $\varphi : \psi \leq b$ and set of constraints Seed, where $b \in \mathbb{Z}$. We define*

$$\begin{aligned}
\textsc{Diff}_{i,\sim}(\varphi) &\triangleq \textsc{Shift}_i(\psi) \sim \psi \\
\textsc{Diff}_{i,\sim}(\mathit{Seed}) &\triangleq \{\textsc{Diff}_{i,\sim}(\varphi) \mid \varphi \in \mathit{Seed}\}
\end{aligned}$$

where $\sim$ is one of $\{>, \leq\}$.

Now we give two procedures Main and DRR (for "Disjunctive Ranking Relation"). DRR, described in Figure 4, is a recursive procedure, that given an LSL L returns a disjunctive ranking relation T such that $R(L) \subseteq T$. Procedure Main, described in Figure 3, repeatedly calls DRR while $R^+(L) \nsubseteq T$, each time feeding DRR with an unrolling of the original L.

```
procedure Main
input: LSL L_original = ⟨COND, UPDATE, 0, 1⟩
output: disjunction ranking relation T or "fail"
begin
   L ← L_original
   T ← ∅
   do
      if DRR(L,COND_L) succeeds with disjunctive ranking relation T'
         T ← T ∪ T'
      else
         return "fail"
   while (binary reachability check on (L_original, T) fails with updated L)
   return T
end.
```

Fig. 3. Procedure Main

Suppose that DRR is called recursively k times with inputs $(L_1, Seed_1)$, $(L_2, Seed_2)$, $\ldots$, $(L_k, Seed_k)$. If the linear ranking function synthesis for L_k succeeds and return the parameters r, δ_0, δ the following is the ranking function we use,

$$\rho_k(X) = \begin{cases} r(X) & \text{if exists } X^1 \text{ such that} \\ & (X, X^1) \in R(L_k) \text{ [case-0]} \\ \delta_0 - \delta & \text{else if exists } X^1 \text{ such that} \\ & (X, X^1) \in R(L_{k-1}) \text{ [case-1]} \\ \delta_0 - 2\delta & \text{else if exists } X^1 \text{ such that} \\ & (X, X^1) \in R(L_{k-2}) \text{ [case-2]} \\ \vdots & \vdots \\ \delta_0 - (k-1)\delta & \text{else if exists } X^1 \text{ such that} \\ & (X, X^1) \in R(L_1) \text{ [case-(k-1)]} \\ \delta_0 - k\delta & \text{otherwise [case-k]} \end{cases} \quad (\sharp)$$

4.3 Correctness Proof

Theorem 2 insures the disjunctive ranking relation returned by DRR is large enough to contain the transition relation of the input LSL. This, in turn, insures that BRC will give a new counterexample for each iteration (until $R^+(L) \subseteq T$)

procedure DRR
input: LSL $L = \langle \textsc{Cond}, \textsc{Update}, 0, j \rangle$, and *Seed* a subset of $\textsc{Cond} \cup \textsc{Update}$.
output: disjunction ranking relation T
begin
  if linear ranking function synthesis on L succeeds with function r
    return $\tau(r)$
  $L_{unroll} \leftarrow \textsc{Unroll}_2(L)$
  for each $\varphi \in Seed$
    $Seed_{triv} \leftarrow \{\textsc{Diff}_{j,>}(\varphi)\}$
    $L_{triv} \leftarrow \langle \textsc{Cond}_{L_{unroll}}, \textsc{Update}_{L_{unroll}} \cup Seed_{triv}, 0, j \rangle$
    $T \leftarrow T \cup \text{DRR}(L_{triv}, Seed_{triv})$
  $Seed \leftarrow \textsc{Diff}_{j,\leq}(Seed)$
  $L \leftarrow \langle \textsc{Cond}_{L_{unroll}}, \textsc{Update}_{L_{unroll}} \cup Seed, 0, j \rangle$
  return $T \cup \text{DRR}(L, Seed)$
end.

Fig. 4. Procedure DRR

and the termination condition converges towards a solution. The proof of theorem 2 relies on lemma 1. Lastly theorem 3 asserts the correctness of the algorithm.

Lemma 1. *Let $s_0, \ldots, s_j, \ldots, s_{m \cdot j}$ be an $(m \cdot j + 1)$-trace of some $L = \langle \textsc{Cond}, \textsc{Update}, 0, j \rangle$ and let Seed be over $X^0, \ldots, X^{m \cdot (j-1)}$. If DRR is called with L and Seed as input and it succeeds then (s_0, s_j) is contained in the return set of DRR.*

Theorem 2. *Suppose that DRR is called with input ($L = \langle \textsc{Cond}, \textsc{Update}, 0, j \rangle$, Seed) where L is over $X^0, \ldots, X^j$ and $Seed = \textsc{Cond}_L$. If DRR terminates successfully with return value T then $R(L) \subseteq T$.*

Theorem 3. *If procedure Main terminates successfully on a program P, then P terminates and has a disjunctive linear ranking relation T.*

4.4 Termination and Complexity of the Algorithm

The procedures Main and DRR as given in this section may not always terminate, in particular when the input LSL is not terminating. When implemented we need to bound the recursion depth of DRR and the number of iterations of the main loop. When the input LSL is deterministic and the variables range over $\mathbb{Z}$, the recursive calls to DRR for each $\varphi \in Seed$ will succeed with no further calls and therefore the number of calls to DRR will be linear in the depth bound. When the LSL is non-deterministic or the variables range over $\mathbb{Q}$ or $\mathbb{R}$, the number of calls to DRR in the worst case is exponential in the depth bound. Finally we note that the LSL $\textsc{Unroll}_k(L)$ has k times as many constraints and variables as in L.

Table 1. Experiment results

#	Vars	Terminating	Linear	Polyrank	Ours	BRC	DRR	Failed Proc
1	1	yes	no	no	no	-	-	DRR
2	1	yes	yes	yes	yes	0	1	-
3	1	yes	yes	yes	yes	0	1	-
4	1	yes	yes	yes	yes	0	1	-
5	1	yes	yes	no	yes	0	1	-
6	2	yes	no	no	yes	0	2	-
7	2	no	-	-	-	-	-	DRR
8	2	no	-	-	-	-	-	DRR
9	2	no	-	-	-	-	-	DRR
10	2	no	-	-	-	-	-	DRR
11	2	yes	no	no	no	-	-	DRR
12	2	yes	no	no	yes	0	2	-
13	2	yes	no	no	yes	0	2	-
14	2	yes	no	no	yes	0	2	-
15	2	yes	yes	no	yes	0	1	-
16	2	no	-	-	-	-	-	DRR
17	2	no	-	-	-	-	-	DRR
18	2	yes	no	no	yes	0	2	-
19	2	no	-	-	-	-	-	DRR
20	2	no	-	-	-	-	-	DRR
21	2	yes	no	no	yes	0	2	-
22	2	no	-	-	-	-	-	DRR
23	2	yes	no	no	yes	0	2	-
24	2	yes	no	no	yes	0	2	-
25	2	yes	yes	yes	yes	0	1	-
26	2	yes	no	no	yes	0	2	-
27	2	yes	no	no	yes	0	2	-
28	3	yes	no	no	yes	0	2	-
29	3	no	-	-	-	-	-	DRR
30	3	yes	no	no	no	∞	3	BRC
31	3	yes	no	no	yes	0	2	-
32	3	yes	no	no	yes	0	2	-
33	3	no	-	-	-	-	-	DRR
34	3	yes	no	no	yes	1	3	-
35	3	yes	no	no	yes	0	2	-
36	3	yes	no	no	yes	0	2	-
37	3	yes	yes	yes	yes	0	1	-
38	4	yes	no	no	yes	0	2	-

5 Experiments

We created a test suite of LSL loops. To our knowledge it is the first LSL test suite. The loops are collected from other research work [27, 15, 34, 28, 9, 12, 6, 8,

13, 5] and real code. The test suite is still growing. At the time of our submission, it contains 38 LSL loops. Among them 11 are non-terminating loops, 7 are terminating with linear ranking functions, 20 are terminating with non-linear ranking functions. Moreover, 6 are non-deterministic, 32 are deterministic, 5 have 1 variable, 22 have 2 variables, 10 have 3 variables, and one has 4 variables. All loops are executed over domain $\mathbb{Z}$. The test suite as well as the implementation are available at [11].

We compared our method to linear ranking function synthesis method [27] using the implementation found in [32], and the polyranking method [7] using the implementation found in [5]. Detailed experimental results are provided in Table 1. The "Vars" column indicates the number of variables used in the LSL. The "Terminating" column indicate whether the LSL terminates. The columns of "Linear", "Polyrank", and "Ours" indicate whether the methods of Podelski *et al.*'s linear ranking function synthesis method [27], Bradley *et al.*'s polyranking method [7], and our method, respectively, have successfully found a termination proof. The "BRC" column states the number of times procedure BRC was called and the "DRR" column states the accumulative depth of DRR recursion. The "Failed Proc" column indicates which procedure, Main or DRR, failed terminating if the whole process failed to terminate. Since the runtime for all three methods was in the magnitude of a few milliseconds we omitted them from the table.

As shown in the table, our method considerably outperformed the other two methods. We succeed for all 7 loops with a linear ranking function. Out of the 20 terminating loops that have no linear ranking function we are successful for 17. For all non-terminating loops, the execution needs to be manually terminated. Except for one loop, all the proof searches fail in procedure DRR. In comparison, the linear ranking function synthesis method [27] succeeds for all the 7 loops with a linear ranking function; it fails to find a termination proof for all the 20 examples among the rest that were terminating. The polyranking method [7] succeeds in proving termination for 5 out of the 7 examples with a linear ranking function; it fails to find a termination proof for all the 20 examples among the rest that were terminating. We set the tree depth to be 100 for the polyranking method.

6 Conclusions

This paper describes an automatic method for generating disjunctive ranking relations for Linear Simple Loops. The method repeatedly finds linear ranking functions on restricted state space until it reaches an over-approximation of the transitive closure of the transition relation. As demonstrated experimentally we largely expanded the scope of LSLs that can be solved. We also extended an existing technique for linear ranking function synthesis. The extended method can handle more general form of LSLs. Another contribution is that we created the first LSL test suite.

Acknowledgments. We thank Byron Cook for providing inspiring examples. We thank the anonymous reviewers for their valuable insights.

References

[1] Balaban, I., Cohen, A., Pnueli, A.: Ranking Abstraction of Recursive Programs. In: Emerson, E.A., Namjoshi, K.S. (eds.) VMCAI 2006. LNCS, vol. 3855, pp. 267–281. Springer, Heidelberg (2005)

[2] Ben-Amram, A.M., Genaim, S., Masud, A.N.: On the Termination of Integer Loops. In: Kuncak, V., Rybalchenko, A. (eds.) VMCAI 2012. LNCS, vol. 7148, pp. 72–87. Springer, Heidelberg (2012)

[3] Berdine, J., Chawdhary, A., Cook, B., Distefano, D., O'Hearn, P.: Variance analyses from invariance analyses. In: Proceedings of the 34th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2007, pp. 211–224. ACM, New York (2007)

[4] Berdine, J., Cook, B., Distefano, D., O'Hearn, P.W.: Automatic Termination Proofs for Programs with Shape-Shifting Heaps. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 386–400. Springer, Heidelberg (2006)

[5] Bradley, A.R.: polyrank: Tools for termination analysis (2005), `http://theory.stanford.edu/~arbrad/software/polyrank.html`

[6] Bradley, A.R., Manna, Z., Sipma, H.B.: Linear Ranking with Reachability. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 491–504. Springer, Heidelberg (2005)

[7] Bradley, A.R., Manna, Z., Sipma, H.B.: The Polyranking Principle. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 1349–1361. Springer, Heidelberg (2005)

[8] Bradley, A.R., Manna, Z., Sipma, H.B.: Termination of Polynomial Programs. In: Cousot, R. (ed.) VMCAI 2005. LNCS, vol. 3385, pp. 113–129. Springer, Heidelberg (2005)

[9] 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)

[10] Chawdhary, A., Cook, B., Gulwani, S., Sagiv, M., Yang, H.: Ranking abstractions. Technical report (2008)

[11] Chen, H.Y., Flur, S., Mukhopadhyay, S.: Lsl test suite, `https://tigerbytes2.lsu.edu/users/hchen11/lsl/`

[12] Colon, M.A., Uribe, T.E.: Generating Finite-State Abstractions of Reactive Systems Using Decision Procedures. In: Hu, A.J., Vardi, M.Y. (eds.) CAV 1998. LNCS, vol. 1427, pp. 293–304. Springer, Heidelberg (1998)

[13] Colón, M.A., Sipma, H.B.: Synthesis of Linear Ranking Functions. In: Margaria, T., Yi, W. (eds.) TACAS 2001. LNCS, vol. 2031, pp. 67–81. Springer, Heidelberg (2001)

[14] Colón, M.A., Sipma, H.B.: Practical Methods for Proving Program Termination. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 442–454. Springer, Heidelberg (2002)

[15] Cook, B., Gulwani, S., Lev-Ami, T., Rybalchenko, A., Sagiv, M.: Proving Conditional Termination. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 328–340. Springer, Heidelberg (2008)

[16] Cook, B., Podelski, A., Rybalchenko, A.: Abstraction Refinement for Termination. In: Hankin, C., Siveroni, I. (eds.) SAS 2005. LNCS, vol. 3672, pp. 87–101. Springer, Heidelberg (2005)

[17] Cook, B., Podelski, A., Rybalchenko, A.: Termination proofs for systems code. In: Proceedings of the 2006 ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2006, pp. 415–426. ACM, New York (2006)
[18] Cook, B., Rybalchenko, A.: Proving that programs eventually do something good. In: POPL 2006: Principles of Programming Languages, pp. 265–276. Springer (2007)
[19] Cousot, P.: Semantic foundations of program analysis. In: Muchnick, S.S., Jones, N.D. (eds.) Program Flow Analysis: Theory and Applications, ch. 10, pp. 303–342. Prentice-Hall, Inc., Englewood Cliffs (1981)
[20] Cousot, P.: Proving Program Invariance and Termination by Parametric Abstraction, Lagrangian Relaxation and Semidefinite Programming. In: Cousot, R. (ed.) VMCAI 2005. LNCS, vol. 3385, pp. 1–24. Springer, Heidelberg (2005)
[21] Cousot, P., Cousot, R.: An abstract interpretation framework for termination. In: POPL, pp. 245–258 (2012)
[22] Gulwani, S., Jain, S., Koskinen, E.: Control-flow refinement and progress invariants for bound analysis. SIGPLAN Not. 44, 375–385 (2009)
[23] Gulwani, S., Mehra, K.K., Chilimbi, T.: Speed: precise and efficient static estimation of program computational complexity. In: Proceedings of the 36th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2009, pp. 127–139. ACM, New York (2009)
[24] Gulwani, S., Zuleger, F.: The reachability-bound problem. In: Proceedings of the 2010 ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2010, pp. 292–304. ACM, New York (2010)
[25] Kroening, D., Sharygina, N., Tsitovich, A., Wintersteiger, C.M.: Termination Analysis with Compositional Transition Invariants. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 89–103. Springer, Heidelberg (2010)
[26] Podelski, A., Rybalchenko, A.: Software model checking of liveness properties via transition invariants. Technical report (2003)
[27] 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)
[28] Podelski, A., Rybalchenko, A.: Transition invariants. In: LICS, pp. 32–41 (2004)
[29] Podelski, A., Rybalchenko, A.: Transition predicate abstraction and fair termination. In: Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2005. ACM (2005)
[30] Ramsey, F.P.: On a problem of formal logic. Proc. London Math. Soc. 30, 491–504 (1930)
[31] Rival, X., Mauborgne, L.: The trace partitioning abstract domain. ACM Trans. Program. Lang. Syst. 29(5) (August 2007)
[32] Rybalchenko, A.: Rankfinder, `http://www.mpi-sws.org/~rybal/rankfinder/`
[33] Schrijver, A.: Theory of linear and integer programming. John Wiley & Sons, Inc., New York (1986)
[34] Tiwari, A.: Termination of Linear Programs. In: Alur, R., Peled, D.A. (eds.) CAV 2004. LNCS, vol. 3114, pp. 70–82. Springer, Heidelberg (2004)
[35] Turing, A.M.: Checking a large routine. Report of a Conference on High Speed Automatic Calculating Machines, pp. 67–69 (1948)