# A Term Rewriting Approach to the Automated Termination Analysis of Imperative Programs*

Stephan Falke and Deepak Kapur

CS Department, University of New Mexico, Albuquerque, NM, USA
{spf, kapur}@cs.unm.edu

**Abstract.** An approach based on term rewriting techniques for the automated termination analysis of imperative programs operating on integers is presented. An imperative programs is transformed into rewrite rules with constraints from quantifier-free Presburger arithmetic. Any computation in the imperative program corresponds to a rewrite sequence, and termination of the rewrite system thus implies termination of the imperative program. Termination of the rewrite system is analyzed using a decision procedure for Presburger arithmetic that identifies possible chains of rewrite rules, and automatically generated polynomial interpretations are used to show finiteness of such chains. An implementation of the approach has been evaluated on a large collection of imperative programs, thus demonstrating its effectiveness and practicality.

## 1 Introduction

Methods for automatically proving termination of imperative programs operating on integers have received increased attention recently. The most commonly used automatic method for this is based on linear ranking functions which linearly combine the values of the program variables in a given state [2, 5, 6, 23, 24]. It was shown in [26] that termination of a simple class of linear programs consisting of a single loop is decidable. More recently, the combination of abstraction refinement and linear ranking functions has been considered [8, 9, 4]. The tool `Terminator` [10], developed at Microsoft Research and based on this idea, has reportedly been used for showing termination of device drivers.

On the other hand, termination analysis for term rewrite systems (TRSs) has been investigated extensively [27]. In this paper, techniques partially based on ideas from the term rewriting literature are used in order to show termination of imperative programs operating on integers. This is done by first translating imperative programs into constrained term rewrite systems based on Presburger arithmetic ($\mathcal{PA}$-based TRS), where the constraints are relations on program variables expressed as quantifier-free formulas from Presburger arithmetic. This way, every computation of the imperative program can be simulated by a rewrite sequence, and termination of the $\mathcal{PA}$-based TRS implies termination of the imperative program.

* Partially supported by NSF grants CCF-0541315 and CNS-0831462.

*Example 1.* The following is an imperative program and its translation into a $\mathcal{PA}$-based TRS.

```
while (x < y) {
   x++
}
```

$$\mathsf{eval}(x, y) \to \mathsf{eval}(x + 1, y)\ [\![x < y]\!]$$

The rule simulates the state change during a single execution of the loop body, and the constraint of the rule corresponds to the condition of the loop. ◊

It is then shown that a $\mathcal{PA}$-based TRS is terminating if and only if it does not admit infinite chains built from the rewrite rules. In order to show absence of infinite chains, termination processors are introduced. Here, a termination processor transform a "complex" termination problem into a set of "simpler" termination problems. This paper presents several such termination processors for $\mathcal{PA}$-based TRSs that are partially based on ideas from the term rewriting literature [20, 1, 13]. The first termination processor uses a decision procedure for Presburger arithmetic to identify possible chains. For this, it is determined which rules from a $\mathcal{PA}$-based TRS may follow each other in a chain. Once possible chains are identified, well-founded relations based on polynomial interpretations are used to show that these chains are finite. Another termination processor can be used to combine $\mathcal{PA}$-based rewrite rules that occur after each other in chains. In particular, the constraints of these two rewrite rules are propagated.

The approach has been implemented in the prototype termination tool `pasta`. An empirical evaluation on a collection of examples taken from recent papers on the termination analysis of imperative programs [2–6, 8, 9, 23, 24] clearly shows the effectiveness and practicality of the method. The most non-trivial part of an implementation is the automatic generation of well-founded relations using polynomial interpretations [20]. The main novelty is that the constraints of the $\mathcal{PA}$-based rewrite rules need to be taken into consideration. Since current methods developed in the term rewriting literature [7, 14] do not support constraints, the development of a new method becomes necessary.

The paper is organized as follows. Section 2 introduces $\mathcal{PA}$-based TRSs. The translation of imperative programs into $\mathcal{PA}$-based TRSs is discussed in Section 3. Next, Section 4 introduces chains and shows that a $\mathcal{PA}$-based TRS is terminating iff it does not admit infinite chains. Furthermore, a framework for showing termination by transforming a $\mathcal{PA}$-based TRS into a set of simpler $\mathcal{PA}$-based TRSs using termination processors is introduced. Section 5 discusses a termination processors that uses a decision procedure for Presburger arithmetic to identify possible chains. Well-founded relations based on polynomial interpretations are introduced in Section 6. Finally, a termination processor that combines $\mathcal{PA}$-based rewrite rules and propagates their constraints is given in Section 7. Section 8 outlines the prototype implementation `pasta` based on the proposed approach. This includes a method for the automatic generation of polynomial interpretations as discussed above. Section 9 concludes and presents an empirical evaluation of `pasta`.

## 2 $\mathcal{PA}$-Based TRSs

In order to model integers, the function symbols from $\mathcal{F}_{\mathcal{PA}} = \{\mathsf{0}, \mathsf{1}, +, -\}$ with types $\mathsf{0}, \mathsf{1} : \mathtt{int}$, $+ : \mathtt{int} \times \mathtt{int} \to \mathtt{int}$, and $- : \mathtt{int} \to \mathtt{int}$ are used. Terms built from these function symbols and a disjoint set $\mathcal{V}$ of variables are called $\mathcal{PA}$-*terms*. This paper uses a simplified, more natural notation for $\mathcal{PA}$-terms, i.e., the $\mathcal{PA}$-term $(x + (-(y + y))) + (\mathsf{1} + (\mathsf{1} + \mathsf{1}))$ will be written as $x - 2y + \mathsf{3}$.

Then, $\mathcal{F}_{\mathcal{PA}}$ is extended by finitely many function symbols $f$ with types $\mathtt{int} \times \ldots \times \mathtt{int} \to \mathtt{univ}$, where $\mathtt{univ}$ is a type distinct from $\mathtt{int}$. The set containing these function symbols is denoted by $\mathcal{F}$, and $\mathcal{T}(\mathcal{F}, \mathcal{F}_{\mathcal{PA}}, \mathcal{V})$ denotes the set of terms of the form $f(s_1, \ldots, s_n)$ where $f \in \mathcal{F}$ and $s_1, \ldots, s_n$ are $\mathcal{PA}$-terms. Notice that nesting of function symbols from $\mathcal{F}$ is *not permitted*, thus resulting in a very simple term structure. This simple structure is none the less sufficient for modeling imperative programs. In the following, $s^*$ denotes a tuple of $\mathcal{PA}$-terms, and notions from terms are extended to tuples of terms component-wise. A *substitution* is a mapping from variables to $\mathcal{PA}$-terms.

**Definition 2 (Syntax of $\mathcal{PA}$-constraints).** *An* atomic $\mathcal{PA}$-constraint *has the form $s \simeq t$, $s \geq t$, or $s > t$ for $\mathcal{PA}$-terms $s, t$. The set of* $\mathcal{PA}$-constraints *is inductively defined as follows:*

1. *$\top$ is a $\mathcal{PA}$-constraint.*
2. *Every atomic $\mathcal{PA}$-constraint is a $\mathcal{PA}$-constraint.*
3. *If $C$ is a $\mathcal{PA}$-constraint, then $\neg C$ is a $\mathcal{PA}$-constraint.*
4. *If $C_1, C_2$ are $\mathcal{PA}$-constraints, then $C_1 \wedge C_2$ is a $\mathcal{PA}$-constraint.*

The Boolean connectives $\vee$, $\Rightarrow$, and $\Leftrightarrow$ are defined as usual. $\mathcal{PA}$-constraints of the form $s < t$ and $s \leq t$ are used to stand for $t > s$ and $t \geq s$, respectively. Also, $s \not\simeq t$ stands for $\neg(s \simeq t)$, and similarly for the other predicates.

$\mathcal{PA}$-constraints have the expected semantics. In the next definition, $\overline{n}$ denotes the integer corresponding to the variable-free $\mathcal{PA}$-term $n$.

**Definition 3 (Semantics of $\mathcal{PA}$-constraints).** *A variable-free $\mathcal{PA}$-constraint $C$ is* $\mathcal{PA}$-valid *iff*

1. *$C$ has the form $\top$, or*
2. *$C$ has the form $s \simeq t$ and $\overline{s} = \overline{t}$ in $\mathbb{Z}$, or*
3. *$C$ has the form $s > t$ and $\overline{s} > \overline{t}$ in $\mathbb{Z}$, or*
4. *$C$ has the form $\neg C_1$ and $C_1$ is not $\mathcal{PA}$-valid, or*
5. *$C$ has the form $C_1 \wedge C_2$ and both $C_1$ and $C_2$ are $\mathcal{PA}$-valid.*

*A $\mathcal{PA}$-constraint $C$ with variables is* $\mathcal{PA}$-valid *iff $C\sigma$ is $\mathcal{PA}$-valid for all ground substitutions $\sigma : \mathcal{V}(C) \to \mathcal{T}(\mathcal{F}_{\mathcal{PA}})$. A $\mathcal{PA}$-constraint $C$ is* $\mathcal{PA}$-satisfiable *iff there exists a ground substitution $\sigma : \mathcal{V}(C) \to \mathcal{T}(\mathcal{F}_{\mathcal{PA}})$ such that $C\sigma$ is $\mathcal{PA}$-valid. Otherwise, $C$ is* $\mathcal{PA}$-unsatisfiable.

$\mathcal{PA}$-validity and $\mathcal{PA}$-satisfiability are decidable [25]. For $\mathcal{PA}$-terms $s, t$, writing $s \simeq_{\mathcal{PA}} t$ is a shorthand for "$s \simeq t$ is $\mathcal{PA}$-valid". Similarly, for terms $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{F}_{\mathcal{PA}}, \mathcal{V})$, $s \simeq_{\mathcal{PA}} t$ iff $s = f(s^*)$ and $t = g(t^*)$ such that $f = g$ and $s^* \simeq_{\mathcal{PA}} t^*$.

The rewrite rules of $\mathcal{PA}$-based TRSs relate terms from $\mathcal{T}(\mathcal{F}, \mathcal{F}_{\mathcal{PA}}, \mathcal{V})$ and are equipped with a $\mathcal{PA}$-constraint. This constraint is used the restrict applicability of the rewrite rule, see Definition 5.

**Definition 4 ($\mathcal{PA}$-Based Rewrite Rules).** *A $\mathcal{PA}$-based rewrite rule has the form $l \to r[\![C]\!]$ where $l, r \in \mathcal{T}(\mathcal{F}, \mathcal{F}_{\mathcal{PA}}, \mathcal{V})$ and $C$ is a $\mathcal{PA}$-constraint. Thus, $l$ and $r$ have the form $f(s_1, \ldots, s_n)$ where $f \in \mathcal{F}$ and $s_1, \ldots, s_n$ are $\mathcal{PA}$-terms.*

The constraint $\top$ is omitted in a $\mathcal{PA}$-based rewrite rule $l \to r[\![\top]\!]$. A *$\mathcal{PA}$-based term rewrite system ($\mathcal{PA}$-based TRS)* $\mathcal{R}$ is a finite set of $\mathcal{PA}$-based rewrite rules. $\mathcal{PA}$-based TRSs give rise to the following rewrite relation. It is based on extended rewriting [22] and requires that the constraint of the $\mathcal{PA}$-based rewrite rule is $\mathcal{PA}$-valid after being instantiated by the matching substitution. Notice that reductions are only possible at the root position of a term and that all variables are instantiated to $\mathcal{PA}$-terms by the substitution.

**Definition 5 (Rewrite Relation).** *For a $\mathcal{PA}$-based TRS $\mathcal{R}$, let $s \to_{\mathcal{PA}\setminus\mathcal{R}} t$ iff there exist a rule $l \to r[\![C]\!] \in \mathcal{R}$ and a substitution $\sigma$ such that*

1. *$s \simeq_{\mathcal{PA}} l\sigma$,*
2. *$C\sigma$ is $\mathcal{PA}$-valid, and*
3. *$t = r\sigma$.*

*Example 6.* Using the $\mathcal{PA}$-based TRS from Example 1, the term $\mathsf{eval}(-1, 1)$ can be reduced using the substitution $\sigma = \{x \mapsto -1, y \mapsto 1\}$ since $\mathsf{eval}(-1, 1) = \mathsf{eval}(x, y)\sigma$ and $(x < y)\sigma = (-1 < 1)$ is $\mathcal{PA}$-valid. Therefore, $\mathsf{eval}(-1, 1) \to_{\mathcal{PA}\setminus\mathcal{R}} \mathsf{eval}(-1 + 1, 1)$. The term $\mathsf{eval}(-1 + 1, 1)$ can be reduced once more, resulting in $\mathsf{eval}(-1 + 1 + 1, 1)$, which cannot be reduced anymore. ◊

## 3 Translating Imperative Programs into $\mathcal{PA}$-based TRSs

In this paper, a simple imperative programming language where programs are formed according to the grammar in Figure 1 is considered. Most constructs in this programming language have the expected meaning, i.e., `skip` is a do-nothing statement, `break` aborts execution of the innermost `while`-loop surrounding it, and `continue` aborts the current iteration of the innermost `while`-loop surrounding it and immediately starts the next iteration. The either- and nondet-constructs can be used to abstract certain aspects of a program that do not need to be modeled precisely. For this, the `either`-statement denotes a nondeterministic choice. The nondet-construct used in arithmetic expressions stands of a nondeterministically chosen value, where nondet-cond can be used to put some constraints on this value. For example, the effect of `x := ?[? < x];` is to assign a new value to the variable `x` that is strictly smaller than the current value of `x`. The assume-statement is used to state preconditions. For the $\mathcal{PA}$-constraints in cond, conjunction is written as `&&`, disjunction is written as `||`, and negation is written as `!`. Furthermore, the predicates are written in their plain text representation. It is assumed that every parallel assignment statement contains each

variable of the program at most once on its left-hand side. A parallel assignment statement `(x1,...,xk) := (e1,...,ek)` with $k = 1$ is also written `x1 := e1`, and `x++` abbreviates `x := x + 1`. Similarly, `x--` abbreviates `x := x - 1`.

```
      prog ::= stmt
             | assume; stmt
      stmt ::= skip
             | assign
             | stmt; stmt
             | if (cond) {stmt} else {stmt}
             | while (cond) {stmt}
             | break
             | continue
             | either {stmt} or {stmt}
    assume ::= assume cond
      cond ::= "PA-constraints"
    assign ::= (var1, ..., vark) := (exp1, ..., expk)    for some k ≥ 1
       exp ::= exp'
             | exp' / nat
      exp' ::= "linear arithmetic expressions, possibly containing nondet"
       nat ::= n    for some n ∈ ℕ − {0}
    nondet ::= ?[nondet-cond]
nondet-cond ::= "PA-constraints using ? as an extra variable"
```

**Fig. 1.** Grammar for the imperative programming language.

The translation now proceeds as follows, where it is assumed that the program uses the variables $\mathtt{x}_1, \ldots, \mathtt{x}_n$. Assume that the program contains $m$ control points (i.e., program entry, `while`-loops and `if`-statements[1]). Then the $i^{\text{th}}$ control point in the program is assigned a function symbol $\mathsf{eval}_i : \mathtt{int} \times \ldots \times \mathtt{int} \to \mathtt{univ}$ with $n$ arguments. For simplicity and without loss of generality, assume that each straight-line code segment between control points is a single parallel assignment statement, `skip`, or empty.

For all $1 \leq i, j \leq m$ such that the $j^{\text{th}}$ control point can be reached from the $i^{\text{th}}$ control point by a straight-line code segment, each such straight-line code segment gives rise to a $\mathcal{PA}$-based rewrite rule of the form

$$\mathsf{eval}_i(\ldots) \to \mathsf{eval}_j(\ldots)[\![C]\!]$$

where the constraint $C$ is determined as follows. If the $i^{\text{th}}$ control point is the program entry, then $C$ is the condition of the `assume`-statement, if any. If the $i^{\text{th}}$ control point is a `while`-loop, then $C$ is the condition of the `while`-loop or the negated condition of the `while`-loop, depending on whether the loop body is entered to reach the $j^{\text{th}}$ control point or not. If the $i^{\text{th}}$ control point is an `if`-statement, then $C$ is the condition of the `if`-statement or the negated condition

[1] For termination purposes it is not necessary to consider the program exit.

of the `if`-statement, depending on whether the then-branch or the else-branch is taken to reach the $j^{\text{th}}$ control point.[2]

If the straight-line code segment is a `skip`-statement or empty, then the rewrite rule becomes

$$\mathsf{eval}_i(x_1, \ldots, x_n) \to \mathsf{eval}_j(x_1, \ldots, x_n)[\![C]\!]$$

If the straight-line code segment is a parallel assignment statement `(`$\mathtt{x}_1$`,...,`$\mathtt{x}_k$`) := (`$e_1$`,...,`$e_k$`)` that does not contain divisions, then the rewrite rule becomes

$$\mathsf{eval}_i(x_1, \ldots, x_n) \to \mathsf{eval}_j(e'_1, \ldots, e'_n)[\![C \wedge D]\!]$$

Here, $e'_i$ is obtained from $e_i$ by replacing each occurrence of an expression of the form `?[`$\mathtt{D}_j$`]` by a fresh variable $z_j$. Furthermore, the constraint $D_j\{\mathtt{?} \mapsto z_j\}$ is added as a conjunct to $D$, where "$\{\mathtt{?} \mapsto z_j\}$ means that the symbol "`?`" for the nondeterministically chosen value is replaced by the fresh variable $z_j$.

Notice that the arguments to $\mathsf{eval}_i$ on the left-hand side are thus distinct variables.

*Example 7.* Using the translation given so far, the imperative program fragment

```
while (x > 0 && y > 0) {
   if (x > y) {
      while (x > 0) {
        x--;
        y++
      }
   } else {
      while (y > 0) {
         y--;
         x++
      }
   }
}
```

is translated into the $\mathcal{PA}$-based rewrite rules

$$\mathsf{eval}_1(x, y) \to \mathsf{eval}_2(x, y) \qquad [\![x > 0 \wedge y > 0 \wedge x > y]\!] \tag{1}$$
$$\mathsf{eval}_1(x, y) \to \mathsf{eval}_3(x, y) \qquad [\![x > 0 \wedge y > 0 \wedge x \not> y]\!] \tag{2}$$
$$\mathsf{eval}_2(x, y) \to \mathsf{eval}_2(x - 1, y + 1)[\![x > 0]\!] \tag{3}$$
$$\mathsf{eval}_2(x, y) \to \mathsf{eval}_1(x, y) \qquad [\![x \not> 0]\!] \tag{4}$$
$$\mathsf{eval}_3(x, y) \to \mathsf{eval}_3(x + 1, y - 1)[\![y > 0]\!] \tag{5}$$
$$\mathsf{eval}_3(x, y) \to \mathsf{eval}_1(x, y) \qquad [\![y \not> 0]\!] \tag{6}$$

[2] It is also possible to combine the control point of an `if`-statement with its (textually) preceding control point. Then $C$ is the conjunction of the constraints obtained from these two control points.

Here, the outer `while`-loop is the first control point and the inner `while`-loop are the second and third control points, i.e., the optimization mentioned in Footnote 2 has been used. This $\mathcal{PA}$-based TRS is used as a running example. $\Diamond$

Next, it is discussed how parallel assignment statements containing division are handled. For simplicity of presentation, only the case where the parallel assignment statement consists of exactly one assignment is considered. Furthermore, it is assumed that this assignment does not contain nondet-constructs. The method extends to the general case in the obvious way. Thus, assume that the assignment to $\mathtt{x}_i$ has the form

$$\mathtt{x}_i := \frac{e}{d}$$

for a linear polynomial $e$ and $d \in \mathbb{N} - \{0\}$. The result of the division is then given as the unique $y$ satisfying the $\mathcal{PA}$-constraint

$$\mathsf{div}(e, d, y, z) := d \cdot y + z \simeq e \wedge [\ (e \geq 0 \wedge 0 \leq z \wedge z < d) \vee (e < 0 \wedge -d < z \wedge z \leq 0)\ ]$$

for some $z$. Intuitively, $y$ is the quotient and $z$ is the remainder of the division. The rewrite rule thus has the shape

$$\mathsf{eval}_i(\ldots, x_i, \ldots) \to \mathsf{eval}_j(\ldots, y, \ldots)[\![C \wedge \mathsf{div}(e, d, y, z)]\!]$$

*Example 8.* The imperative program

```
while (l < u) {
  either {
    l := (l + u + 2) / 2
  } or {
    u := (l + u) / 2
  }
}
```

is translated into the $\mathcal{PA}$-based rewrite rules

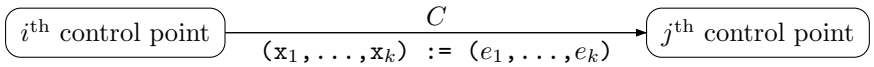
$$\mathsf{eval}(l, u) \to \mathsf{eval}(l', u)[\![l < u \wedge \mathsf{div}(l + u + 2, 2, l', z)]\!]$$
$$\mathsf{eval}(l, u) \to \mathsf{eval}(l, u')[\![l < u \wedge \mathsf{div}(l + u, 2, u', z)]\!]$$

The above imperative program contains the updates to the upper and lower bound used in binary search. $\Diamond$

The following theorem is based on the observation that any state transition of the imperative program can be mimicked by a rewrite sequence.

**Theorem 9.** *Let $P$ be an imperative program. Then the above translation produces a $\mathcal{PA}$-based TRS $\mathcal{R}_P$ such that $P$ is terminating if $\mathcal{R}_P$ is terminating.*

*Proof sketch.* That the translation produces a $\mathcal{PA}$-based TRS is immediate by inspection. For the second statement, consider the control flow graph associated with $P$, where, as in [24], each control point produces a node and the transitions are labeled by the parallel assignment statement executed during that transition and the condition obtained from the `while`-loop, `if`-, or `assume`-statement. A typical transition has the form




It now suffices to notice that the translation produces the rewrite rule

$$\mathsf{eval}_i(\ldots) \to \mathsf{eval}_j(\ldots)[\![C]\!]$$

corresponding to this transition. □

Notice that $\mathcal{R}_P$ might be non-terminating even if $P$ is terminating. The translation can be replaced by a different translation and it is possible to consider translations from more general transition systems as long as the statement of the theorem is satisfied. The remainder of this paper is concerned with methods for showing termination of $\mathcal{PA}$-based TRSs.

## 4 Characterizing Termination of $\mathcal{PA}$-Based TRSs

In order to verify termination of $\mathcal{PA}$-based TRSs, the notion of *chains* is used. Intuitively, a chain represents a possible sequence of rule applications in a reduction w.r.t. $\to_{\mathcal{PA}\setminus\mathcal{R}}$. In the following, it is always assumed that different (occurrences of) $\mathcal{PA}$-based rewrite rules are variable-disjoint, and the domain of substitutions may be infinite. This allows for a single substitution in the following definition. Recall that $\to_{\mathcal{PA}\setminus\mathcal{R}}$ is only applied at the root position of a term.

**Definition 10 ($\mathcal{R}$-Chains).** *Let $\mathcal{R}$ be a $\mathcal{PA}$-based TRS. A (possibly infinite) sequence of $\mathcal{PA}$-based rewrite rules $l_1 \to r_1[\![C_1]\!], l_2 \to r_2[\![C_2]\!], \ldots$ from $\mathcal{R}$ is an* $\mathcal{R}$-chain *iff there exists a substitution $\sigma$ such that $r_i\sigma \simeq_{\mathcal{PA}} l_{i+1}\sigma$ and $C_i\sigma$ is $\mathcal{PA}$-valid for all $i \geq 1$.*

*Example 11.* Continuing Example 7, the $\mathcal{R}$-chain

$$\begin{aligned}
\mathsf{eval}_1(x,y) &\to \mathsf{eval}_2(x,y) && [\![x > 0 \wedge y > 0 \wedge x > y]\!]\\
\mathsf{eval}_2(x',y') &\to \mathsf{eval}_2(x'-1,y'+1) && [\![x' > 0]\!]\\
\mathsf{eval}_2(x'',y'') &\to \mathsf{eval}_2(x''-1,y''+1) && [\![x'' > 0]\!]\\
\mathsf{eval}_2(x''',y''') &\to \mathsf{eval}_1(x''',y''') && [\![x''' \not> 0]\!]
\end{aligned}$$

can be built by considering the substitution $\sigma = \{x \mapsto 2, x' \mapsto 2, x'' \mapsto 1, x''' \mapsto 0, y \mapsto 1, y' \mapsto 1, y'' \mapsto 2, y''' \mapsto 3\}$ since then $\mathsf{eval}_2(x,y)\sigma = \mathsf{eval}_2(2,1) = \mathsf{eval}_2(x',y')\sigma$, $\mathsf{eval}_2(x'-1,y'+1)\sigma = \mathsf{eval}_2(2-1,1+1) \simeq_{\mathcal{PA}} \mathsf{eval}_2(1,2) = \mathsf{eval}_2(x'',y'')\sigma$, and $\mathsf{eval}_2(x''-1,y''+1)\sigma = \mathsf{eval}_2(1-1,2+1) \simeq_{\mathcal{PA}} \mathsf{eval}_2(0,3) = \mathsf{eval}_2(x''',y''')\sigma$, where additionally $(x > 0 \wedge y > 0 \wedge x > y)\sigma = (2 > 0 \wedge 1 > 0 \wedge 2 > 1)$, $(x' > 0)\sigma = (2 > 0)$, $(x'' > 0)\sigma = (1 > 0)$, and $(x''' \not> 0)\sigma = (0 \not> 0)$ are $\mathcal{PA}$-valid. ◊

Using the notion of $\mathcal{R}$-chains, the following characterization of termination of a $\mathcal{PA}$-based TRS $\mathcal{R}$ is immediate.

**Theorem 12.** *Let $\mathcal{R}$ be a $\mathcal{PA}$-based TRS. Then $\mathcal{R}$ is terminating if and only if there are no infinite $\mathcal{R}$-chains.*

*Proof.* Let $\mathcal{R}$ be a $\mathcal{PA}$-based TRS.

"$\Leftarrow$" Assume there exists a term $s \in \mathcal{T}(\mathcal{F}, \mathcal{F}_{\mathcal{PA}}, \mathcal{V})$ which starts an infinite $\to_{\mathcal{PA}\setminus\mathcal{R}}$-reduction and consider an infinite reduction starting with $s$. According to the definition of $\to_{\mathcal{PA}\setminus\mathcal{R}}$, there exist a $\mathcal{PA}$-based rewrite rule $l_1 \to r_1[\![C_1]\!] \in \mathcal{R}$ and a substitution $\sigma_1$ such that $s \simeq_{\mathcal{PA}} l_1\sigma_1$ and $C_1\sigma$ is $\mathcal{PA}$-valid. The reduction then yields $r_1\sigma_1$ and the infinite $\to_{\mathcal{PA}\setminus\mathcal{R}}$-reduction continues with $r_1\sigma_1$, i.e., the term $r_1\sigma_1$ starts an infinite $\to_{\mathcal{PA}\setminus\mathcal{R}}$-reduction as well. The first $\mathcal{PA}$-based rewrite rule in the infinite $\mathcal{R}$-chain that is getting constructed is $l_1 \to r_1[\![C_1]\!]$. The other $\mathcal{PA}$-based rewrite rules of the infinite $\mathcal{R}$-chain are determined in the same way: let $l_i \to r_i[\![C_i]\!]$ be a $\mathcal{PA}$-based rewrite rule such that $r_i\sigma_i$ starts an infinite $\to_{\mathcal{PA}\setminus\mathcal{R}}$-reduction. Again a $\mathcal{PA}$-based rewrite rule $l_{i+1} \to r_{i+1}[\![C_{i+1}]\!]$ is applied to $r_i\sigma_i$ using a substitution $\sigma_{i+1}$ and the term $r_{i+1}\sigma_{i+1}$ starts an infinite $\to_{\mathcal{PA}\setminus\mathcal{R}}$-reduction. This produces the next $\mathcal{PA}$-based rewrite rule in the infinite $\mathcal{R}$-chain. In this way, the infinite sequence

$$l_1 \to r_1[\![C_1]\!],\ l_2 \to r_2[\![C_2]\!],\ l_3 \to r_3[\![C_3]\!],\ \ldots$$

is obtained. Since it is assumed that different (occurrences of) $\mathcal{PA}$-based rewrite rules are variable-disjoint, the substitution $\sigma = \sigma_1 \cup \sigma_2 \cup \ldots$ gives $r_i\sigma \simeq_{\mathcal{PA}} l_{i+1}\sigma$ and $\mathcal{PA}$-validity of the instantiated $\mathcal{PA}$-constraint $C_i\sigma$ for all $i \geq 1$. Thus, the above infinite sequence is indeed an infinite $\mathcal{R}$-chain.

"$\Rightarrow$" Assume there exists an infinite $\mathcal{R}$-chain

$$l_1 \to r_1[\![C_1]\!],\ l_2 \to r_2[\![C_2]\!],\ l_3 \to r_3[\![C_3]\!],\ \ldots$$

Hence, there exists a substitution $\sigma$ such that

$$\begin{aligned} r_1\sigma &\simeq_{\mathcal{PA}} l_2\sigma, \\ r_2\sigma &\simeq_{\mathcal{PA}} l_3\sigma, \\ &\vdots \end{aligned}$$

and the instantiated $\mathcal{PA}$-constraints $C_1\sigma, C_2\sigma, \ldots$ are $\mathcal{PA}$-valid.

From this, the infinite $\to_{\mathcal{PA}\setminus\mathcal{R}}$-reduction

$$r_1\sigma \to_{\mathcal{PA}\setminus\mathcal{R}} r_2\sigma \to_{\mathcal{PA}\setminus\mathcal{R}} r_3\sigma \ldots$$

is obtained, and $\mathcal{R}$ is thus not terminating. □

In the next sections, various techniques for showing termination of $\mathcal{PA}$-based TRSs are developed. These techniques are stated independently of each other in the form of *termination processors*, following the dependency pair framework for ordinary term rewriting [15] and for term rewriting with built-in numbers [13]. The main motivation for this approach is that it allows to combine different termination techniques in a flexible manner since it typically does not suffice to just use a single technique in a successful termination proof.

Termination processors are used to transform a $\mathcal{PA}$-based TRS into a (finite) set of simpler $\mathcal{PA}$-based TRSs for which termination is (hopefully) easier to show. A termination processor $\mathsf{Proc}$ is *sound* iff for all $\mathcal{PA}$-based TRSs $\mathcal{R}$, $\mathcal{R}$ is terminating whenever all $\mathcal{PA}$-based TRSs in $\mathsf{Proc}(\mathcal{R})$ are terminating. Notice that $\mathsf{Proc}(\mathcal{R}) = \{\mathcal{R}\}$ is possible. This can be interpreted as a failure of $\mathsf{Proc}$ and indicates that a different termination processor should be applied.

Using sound termination processors, a termination proof of a $\mathcal{PA}$-based TRS $\mathcal{R}$ then consists of the recursive application of these processors. If all $\mathcal{PA}$-based TRSs obtained in this process are transformed into $\emptyset$, then $\mathcal{R}$ is terminating.

## 5 Termination Graphs

Notice that a $\mathcal{PA}$-based TRS $\mathcal{R}$ might give rise to infinitely many different $\mathcal{R}$-chains. This section introduces a method that represents these infinitely many chains in a finite graph. Then, each $\mathcal{R}$-chain (and thus each computation path in the imperative program) corresponds to a path in this graph. By considering the strongly connected components of this graph, it then becomes possible to decompose a $\mathcal{PA}$-based TRS into several independent $\mathcal{PA}$-based TRSs by determining which $\mathcal{PA}$-based rewrite rules may follow each other in a chain.

The termination processor based on this idea uses *termination graphs*. This notion is motivated by the notion of dependency graphs used in the dependency pair framework for ordinary term rewriting [1] and normalized equational rewriting with constraints [13].

**Definition 13 (Termination Graphs).** *Let $\mathcal{R}$ be a $\mathcal{PA}$-based TRS. The nodes of the* $\mathcal{R}$-termination graph $\mathsf{TG}(\mathcal{R})$ *are the $\mathcal{PA}$-based rewrite rules from $\mathcal{R}$ and there is an arc from $l_1 \to r_1[\![C_1]\!]$ to $l_2 \to r_2[\![C_2]\!]$ iff $l_1 \to r_1[\![C_1]\!]$, $l_2 \to r_2[\![C_2]\!]$ is an $\mathcal{R}$-chain.*

In contrast to [1, 13], it is decidable whether there is an arc from $l_1 \to r_1[\![C_1]\!]$ to $l_2 \to r_2[\![C_2]\!]$. Let $r_1 = f(s^*)$ and $l_2 = g(t^*)$. If $f \neq g$ then there is no arc between the $\mathcal{PA}$-based rewrite rules. Otherwise, there is an arc between the $\mathcal{PA}$-based rewrite rules iff there is a substitution $\sigma$ such that the constraint $s^*\sigma \simeq t^*\sigma \wedge C_1\sigma \wedge C_2\sigma$ is $\mathcal{PA}$-valid, i.e., iff $s^* \simeq t^* \wedge C_1 \wedge C_2$ is $\mathcal{PA}$-satisfiable.

A set $\mathcal{R}' \subseteq \mathcal{R}$ of $\mathcal{PA}$-based rewrite rules is a *cycle* in $\mathsf{TG}(\mathcal{R})$ iff for all $\mathcal{PA}$-based rewrite rules $l_1 \to r_1[\![C_1]\!]$ and $l_2 \to r_2[\![C_2]\!]$ from $\mathcal{R}'$ there exists a non-empty path from $l_1 \to r_1[\![C_1]\!]$ to $l_2 \to r_2[\![C_2]\!]$ that only traverses $\mathcal{PA}$-based rewrite rules from $\mathcal{R}'$. A cycle is a *strongly connected component* (SCC) if it is not a proper subset of any other cycle. Now, every infinite $\mathcal{R}$-chain contains an

infinite tail that stays within a cycle of $\mathsf{TG}(\mathcal{R})$, and it is thus sufficient to prove the absence of infinite chains for each SCC separately.

**Theorem 14 (Processor Based on Termination Graphs).** *The termination processor with* $\mathsf{Proc}(\mathcal{R}) = \{\mathcal{R}_1, \ldots, \mathcal{R}_n\}$*, where* $\mathcal{R}_1, \ldots, \mathcal{R}_n$ *are the SCCs of* $\mathsf{TG}(\mathcal{R})$*, is sound.*[3]

*Proof.* After a finite number of $\mathcal{PA}$-based rewrite rules in the beginning, any infinite $\mathcal{R}$-chain only contains $\mathcal{PA}$-based rewrite rules from some SCC. Hence, every infinite $\mathcal{R}$-chain gives rise to an infinite $\mathcal{R}_i$-chain for some $1 \leq i \leq n$ and $\mathsf{Proc}$ is thus sound. □

*Example 15.* Continuing Example 7, recall the termination problem

$$\mathsf{eval}_1(x, y) \to \mathsf{eval}_2(x, y) \qquad [\![x > 0 \wedge y > 0 \wedge x > y]\!] \tag{1}$$
$$\mathsf{eval}_1(x, y) \to \mathsf{eval}_3(x, y) \qquad [\![x > 0 \wedge y > 0 \wedge x \not> y]\!] \tag{2}$$
$$\mathsf{eval}_2(x, y) \to \mathsf{eval}_2(x - 1, y + 1) [\![x > 0]\!] \tag{3}$$
$$\mathsf{eval}_2(x, y) \to \mathsf{eval}_1(x, y) \qquad [\![x \not> 0]\!] \tag{4}$$
$$\mathsf{eval}_3(x, y) \to \mathsf{eval}_3(x + 1, y - 1) [\![y > 0]\!] \tag{5}$$
$$\mathsf{eval}_3(x, y) \to \mathsf{eval}_1(x, y) \qquad [\![y \not> 0]\!] \tag{6}$$

This termination problem gives rise to the termination graph


The termination graph contains two SCC and the termination processor of Theorem 14 returns the termination problems $\{(3)\}$ and $\{(5)\}$, which can be handled independently of each other. ◊

## 6 $\mathcal{PA}$-Polynomial Interpretations

In this section, well-founded relations on terms are considered and it is shown that $\mathcal{PA}$-based rewrite rules may be deleted from a $\mathcal{PA}$-based TRS if their left-hand side is strictly "bigger" than their right-hand side. A promising way for the generation of such well-founded relations is the use of polynomial interpretations [20]. In contrast to [20], $\mathcal{PA}$-based TRSs allow for the use of polynomial interpretations with coefficients from $\mathbb{Z}$. In the term rewriting literature, polynomial interpretations with coefficients from $\mathbb{Z}$ have been utilized in [17, 16, 13].

A *$\mathcal{PA}$-polynomial interpretation* maps each symbol $f \in \mathcal{F}$ to a polynomial over $\mathbb{Z}$ such that $\mathcal{P}ol(f) \in \mathbb{Z}[x_1, \ldots, x_n]$ if $f$ has $n$ arguments. The mapping $\mathcal{P}ol$ is then extended to terms from $\mathcal{T}(\mathcal{F}, \mathcal{F}_{\mathcal{PA}}, \mathcal{V})$ by letting $[f(t_1, \ldots, t_n)]_{\mathcal{P}ol} =$

[3] Notice, in particular, that $\mathsf{Proc}(\emptyset) = \emptyset$. Also, notice that $\mathcal{PA}$-based rewrite rules with unsatisfiable constraints are not connected to any $\mathcal{PA}$-based rewrite rule and do thus not occur in any SCC.

$\mathcal{P}ol(f)(t_1,\ldots,t_n)$ for all $f \in \mathcal{F}$. Now $\mathcal{PA}$-polynomial interpretations generate relations on terms as follows. Here, the requirement $[s]_{\mathcal{P}ol} \geq 0$ is needed for well-foundedness of $\succ_{\mathcal{P}ol}$.

**Definition 16 ($\succ_{\mathcal{P}ol}$ and $\gtrsim_{\mathcal{P}ol}$).** *Let $\mathcal{P}ol$ be a $\mathcal{PA}$-polynomial interpretation, let $s,t \in \mathcal{T}(\mathcal{F},\mathcal{F}_{\mathcal{PA}},\mathcal{V})$, and let $C$ be a $\mathcal{PA}$-constraint. Then*

- *$s \succ^C_{\mathcal{P}ol} t$ iff $\forall^*.\ C \Rightarrow [s]_{\mathcal{P}ol} \geq 0$ and $\forall^*.\ C \Rightarrow [s]_{\mathcal{P}ol} > [t]_{\mathcal{P}ol}$ are true in the integers, where, $\forall^*$ denotes the universal closure of the formula.*
- *$s \gtrsim^C_{\mathcal{P}ol} t$ iff $\forall^*.\ C \Rightarrow [s]_{\mathcal{P}ol} \geq [t]_{\mathcal{P}ol}$ is true in the integers.*

Using $\mathcal{PA}$-polynomial interpretations, $\mathcal{PA}$-based rewrite rules $l \to r[\![C]\!]$ with $l \succ^C_{\mathcal{P}ol} r$ can be removed from a $\mathcal{PA}$-based TRS if all remaining $\mathcal{PA}$-based rewrite rules $l' \to r'[\![C']\!]$ satisfy $l' \gtrsim^{C'}_{\mathcal{P}ol} r'$.

**Theorem 17 (Processor Based on $\mathcal{PA}$-Polynomial Interpretations).** *Let $\mathcal{P}ol$ be a $\mathcal{PA}$-polynomial interpretation and let* Proc *be the termination processor with* $\mathsf{Proc}(\mathcal{R}) =$

- *$\{\mathcal{R}-\mathcal{R}'\}$, if $\mathcal{R}' \subseteq \mathcal{R}$ such that*
  - *$l \succ^C_{\mathcal{P}ol} r$ for all $l \to r[\![C]\!] \in \mathcal{R}'$, and*
  - *$l \gtrsim^C_{\mathcal{P}ol} r$ for all $l \to r[\![C]\!] \in \mathcal{R}-\mathcal{R}'$.*
- *$\{\mathcal{R}\}$, otherwise.*

*Then* Proc *is sound.*

*Proof.* This is a special case of Theorem 23 from Section 6.1. □

*Example 18.* Continuing Example 15, recall the $\mathcal{PA}$-based TRSs $\{(3)\}$ and $\{(5)\}$ that can be handled independently of each other. For the first $\mathcal{PA}$-based TRS, a $\mathcal{PA}$-polynomial interpretation such that $\mathcal{P}ol(\mathsf{eval}_2) = x_1$ can be used. Then $\mathsf{eval}_2(x,y) \succ^{[\![x>0]\!]}_{\mathcal{P}ol} \mathsf{eval}_2(x-1,y+1)$ since $\forall x.\ x>0 \Rightarrow x \geq 0$ and $\forall x.\ x>0 \Rightarrow x > x-1$ are true in the integers. Applying the termination processor of Theorem 17, the first $\mathcal{PA}$-based TRS is thus transformed into the trivial $\mathcal{PA}$-based TRS $\emptyset$. The second $\mathcal{PA}$-based TRS can be handled similarly using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}_3) = x_2$. ◊

If $\mathcal{P}ol(f)$ is linear for all $f$, then it is decidable whether $l \succ^C_{\mathcal{P}ol} r$ or $l \gtrsim^C_{\mathcal{P}ol} r$ is true. A method for the automatic generation of suitable $\mathcal{PA}$-polynomial interpretations is presented in Section 8.1.

### 6.1 $\mathcal{PA}$-Reduction Pairs

For the proof of Theorem 17, it is convenient to give an abstract characterization of well-founded relations on terms that may be used for termination proofs of $\mathcal{PA}$-based TRSs. The relations that can be used may not make a distinction between terms that are $\mathcal{PA}$-equivalent, i.e., they need to satisfy the following requirement.

**Definition 19 ($\mathcal{PA}$-Compatible Relations).** *A relation $\bowtie$ on $\mathcal{T}(\mathcal{F}, \mathcal{F}_{\mathcal{PA}}, \mathcal{V})$ is* $\mathcal{PA}$-compatible *iff $s' \simeq_{\mathcal{PA}} s \bowtie t \simeq_{\mathcal{PA}} t'$ implies $s' \bowtie t'$ for all $s, t, s', t'$.*

The notion of $\mathcal{PA}$-reduction pairs is motivated by the notion of reduction pairs [19]. A $\mathcal{PA}$-reduction pair consists of two relations $\gtrsim$ and $\succ$, where it is *not* required that $\succ$ is the strict part of $\gtrsim$.

**Definition 20 ($\mathcal{PA}$-Reduction Pairs).** *A* $\mathcal{PA}$-reduction pair $(\gtrsim, \succ)$ *consists of two relations on $\mathcal{T}(\mathcal{F}, \mathcal{F}_{\mathcal{PA}}, \mathcal{V})$ such that $\succ$ is well-founded, $\gtrsim$ and $\succ$ are $\mathcal{PA}$-compatible, and $\succ$ is compatible with $\gtrsim$, i.e., $\gtrsim \circ \succ \subseteq \succ$ or $\succ \circ \gtrsim \subseteq \succ$.*

Relations on $\mathcal{T}(\mathcal{F}, \mathcal{F}_{\mathcal{PA}}, \mathcal{V})$ are extended to operate on terms with constraints as follows. Intuitively, it suffices to consider all instantiations that make the constraint $\mathcal{PA}$-valid.

**Definition 21 (Relations on Constrained Terms).** *Let $\bowtie$ be a relation on $\mathcal{T}(\mathcal{F}, \mathcal{F}_{\mathcal{PA}}, \mathcal{V})$. Let $s, t$ be terms and let $C$ be a $\mathcal{PA}$-constraint. Then $s \bowtie^C t$ iff $s\sigma \bowtie t\sigma$ for all substitutions $\sigma$ such that $C\sigma$ is $\mathcal{PA}$-valid.*

*Example 22.* Consider the relation $>_{\mathcal{PA}}$ on $\mathcal{PA}$-terms over $\mathcal{V}$, defined by $s >_{\mathcal{PA}} t$ iff $s > t$ is $\mathcal{PA}$-valid. Then $x + y \not>_{\mathcal{PA}} x$ since $x + y > x$ is not $\mathcal{PA}$-valid. On the other hand, $x + y >_{\mathcal{PA}}^{[\![y>0]\!]} x$. ◊

Using $\mathcal{PA}$-reduction pairs, $\mathcal{PA}$-based rewrite rules $l \to r[\![C]\!]$ such that $l \succ^C r$ can be removed from a termination problem if all remaining $\mathcal{PA}$-based rewrite rules $l' \to r'[\![C']\!]$ satisfy $l' \gtrsim^{C'} r'$. This generalizes Theorem 17.

**Theorem 23 (Processor Based on $\mathcal{PA}$-Reduction Pairs).** *Let $(\gtrsim, \succ)$ be a $\mathcal{PA}$-reduction pair and let* Proc *be the termination processor with* $\mathsf{Proc}(\mathcal{R}) =$

- $\{\mathcal{R} - \mathcal{R}'\}$, *if $\mathcal{R}' \subseteq \mathcal{R}$ such that*
  - *$l \succ^C r$ for all $l \to r[\![C]\!] \in \mathcal{R}'$, and*
  - *$l \gtrsim^C r$ for all $l \to r[\![C]\!] \in \mathcal{R} - \mathcal{R}'$.*
- $\{\mathcal{R}\}$, *otherwise.*

*Then* Proc *is sound.*

*Proof.* In the second case soundness is obvious. Otherwise, it needs to be shown that every infinite $\mathcal{R}$-chain contains only finitely many $\mathcal{PA}$-based rewrite rules from $\mathcal{R}'$. Thus, assume that $l_1 \to r_1[\![C_1]\!], l_2 \to r_2[\![C_2]\!], \ldots$ is an infinite $\mathcal{R}$-chain using the substitution $\sigma$. Hence, $r_i\sigma \simeq_{\mathcal{PA}} l_{i+1}\sigma$ and $C_i\sigma$ is $\mathcal{PA}$-valid for all $i \geq 1$.

Since $l_i \gtrsim^{C_i} r_i$ for all $l_i \to r_i[\![C_i]\!] \in \mathcal{R} - \mathcal{R}'$ and $l_i \succ^{C_i} r_i$ for all $l_i \to r_i[\![C_i]\!] \in \mathcal{R}'$, this implies $l_i\sigma \gtrsim r_i\sigma$ or $l_i\sigma \succ r_i\sigma$ for all $i \geq 1$. Hence, the infinite $\mathcal{R}$-chain gives rise to

$$l_1\sigma \bowtie_1 r_1\sigma \simeq_{\mathcal{PA}} l_2\sigma \bowtie_2 r_2\sigma \simeq_{\mathcal{PA}} l_3\sigma \ldots$$

where $\bowtie_i \in \{\gtrsim, \succ\}$. Since $\gtrsim$ and $\succ$ are $\mathcal{PA}$-compatible,

$$l_1\sigma \bowtie_1 l_2\sigma \bowtie_2 l_3\sigma \ldots$$

If the infinite $\mathcal{R}$-chain contains infinitely many $\mathcal{PA}$-based rewrite rules from $\mathcal{R}'$, then $\bowtie_i = \succ$ for infinitely many $i$. In this case, the compatibility of $\succ$ with $\gtrsim$ produces an infinite $\succ$-chain, contradicting the well-foundedness of $\succ$. Thus, only finitely many $\mathcal{PA}$-based rewrite rules from $\mathcal{R}'$ occur in the infinite $\mathcal{R}$-chain and there thus exists an infinite $(\mathcal{R} - \mathcal{R}')$-chain as well. □

For the proof of Theorem 17, it thus suffices to show that $\mathcal{PA}$-polynomial interpretations give rise to $\mathcal{PA}$-reduction pairs. For this, $\succ_{\mathcal{P}ol}$ is defined by $s \succ_{\mathcal{P}ol} t$ iff $s \succ_{\mathcal{P}ol}^{[\![\top]\!]} t$, and similarly for $\gtrsim_{\mathcal{P}ol}$.

**Theorem 24.** *Let $\mathcal{P}ol$ be a $\mathcal{PA}$-polynomial interpretation. Then $(\gtrsim_{\mathcal{P}ol}, \succ_{\mathcal{P}ol})$ is a $\mathcal{PA}$-reduction pair.*

*Proof.* It needs to be shown that $\succ_{\mathcal{P}ol}$ is well-founded, that $\gtrsim_{\mathcal{P}ol}$ and $\succ_{\mathcal{P}ol}$ are $\mathcal{PA}$-compatible, and that $\succ_{\mathcal{P}ol}$ is compatible with $\gtrsim_{\mathcal{P}ol}$.

*$\succ_{\mathcal{P}ol}$ is well-founded:* For a contradiction, assume that $s_1 \succ_{\mathcal{P}ol} s_2 \succ_{\mathcal{P}ol} \ldots$ is an infinite descending sequence of terms. This means that $[s_i]_{\mathcal{P}ol} > [s_{i+1}]_{\mathcal{P}ol}$ and $[s_i]_{\mathcal{P}ol} \geq 0$ for all $i \geq 1$ and all instantiations of the variables by integers. By fixing an arbitrary instantiation, integers $d_1, d_2, \ldots \geq 0$ are obtained such that $d_1 > d_2 > \ldots$, which is clearly impossible.

*$\gtrsim_{\mathcal{P}ol}$ and $\succ_{\mathcal{P}ol}$ are $\mathcal{PA}$-compatible.* Let $s \gtrsim_{\mathcal{P}ol} t$ and assume that $s' \simeq_{\mathcal{PA}} s$ and $t \simeq_{\mathcal{PA}} t'$. Then $s = f(s^*), s' = f(s'^*), t = g(t^*)$, and $t' = g(t'^*)$, where $s^* \simeq_{\mathcal{PA}} s'^*$ and $t^* \simeq_{\mathcal{PA}} t'^*$. Clearly, $s \simeq_{\mathcal{PA}} s'$ implies that $s$ and $s'$ are equal for all instantiations of the variables by integers. Thus, $[s']_{\mathcal{P}ol} = \mathcal{P}ol(f)(s'_1, \ldots, s'_n) = \mathcal{P}ol(f)(s_1, \ldots, s_n) \geq \mathcal{P}ol(g)(t_1, \ldots, t_m) = \mathcal{P}ol(g)(t'_1, \ldots, t'_n) = [t']_{\mathcal{P}ol}$ for all instantiations of the variables by integers since $s \gtrsim_{\mathcal{P}ol} t$. But this means $s' \gtrsim_{\mathcal{P}ol} t'$. The $\mathcal{PA}$-compatibility of $\succ_{\mathcal{P}ol}$ is shown the same way.

*$\succ_{\mathcal{P}ol}$ is compatible with $\gtrsim_{\mathcal{P}ol}$:* For showing that $\succ_{\mathcal{P}ol} \circ \gtrsim_{\mathcal{P}ol} \subseteq \succ_{\mathcal{P}ol}$, let $s \succ_{\mathcal{P}ol} t \gtrsim_{\mathcal{P}ol} u$, i.e., $[s]_{\mathcal{P}ol} > [t]_{\mathcal{P}ol} \geq [u]_{\mathcal{P}ol}$ and $[s]_{\mathcal{P}ol} \geq 0$ for all instantiations of the variables by integers. But then $[s]_{\mathcal{P}ol} > [u]_{\mathcal{P}ol}$ for all instantiations of the variables as well and therefore $s \succ_{\mathcal{P}ol} u$.

Also, $\gtrsim_{\mathcal{P}ol} \circ \succ_{\mathcal{P}ol} \subseteq \succ_{\mathcal{P}ol}$. To see this, let $s \gtrsim_{\mathcal{P}ol} t \succ_{\mathcal{P}ol} u$. Then $[s]_{\mathcal{P}ol} \geq [t]_{\mathcal{P}ol} > [u]_{\mathcal{P}ol}$ and $[t]_{\mathcal{P}ol} \geq 0$ for all instantiations of the variables by integers. But then also $[s]_{\mathcal{P}ol} \geq 0$ and $[s]_{\mathcal{P}ol} > [u]_{\mathcal{P}ol}$ for all instantiations of the variables, i.e., $s \succ_{\mathcal{P}ol} u$. □

## 7 Chaining

It is possible to replace a $\mathcal{PA}$-based rewrite rule $l \to r[\![C]\!]$ by a set of new $\mathcal{PA}$-based rewrite rules that are formed by chaining $l \to r[\![C]\!]$ to the $\mathcal{PA}$-based rewrite rules that may follow it in an infinite chain.[4] This way, further information about

[4] Dually, it is possible to consider the $\mathcal{PA}$-based rewrite rules that may *precede* it.

the possible substitutions used for a chain can be obtained. Chaining of $\mathcal{PA}$-based rewrite rules corresponds to executing bigger parts of the imperative program at once, spanning several control points. Within this section, it is assume that all $\mathcal{PA}$-based rewrite rules have the form $f(x_1,\ldots,x_n) \to r[\![C]\!]$, where $x_1,\ldots,x_n$ are distinct variables. Recall that the $\mathcal{PA}$-based rewrite rules generated by the translation from Section 3 satisfy this requirement.

*Example 25.* Consider the following imperative program and the $\mathcal{PA}$-based TRS generated from it.

```
while (x > z) {
   while (y > z) {
      y--
   }
   x--
}
```

$$\mathsf{eval}_1(x,y,z) \to \mathsf{eval}_2(x,y,z)\ [\![x > z]\!] \quad (7)$$
$$\mathsf{eval}_2(x,y,z) \to \mathsf{eval}_2(x,y-1,z)\ [\![y > z]\!] \quad (8)$$
$$\mathsf{eval}_2(x,y,z) \to \mathsf{eval}_1(x-1,y,z)\ [\![y \not> z]\!] \quad (9)$$

The $\mathcal{PA}$-based TRS $\{(7),(8),(9)\}$ is transformed into $\{(7),(9)\}$ using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}_1) = \mathcal{P}ol(\mathsf{eval}_2) = x_2 - x_3$. The $\mathcal{PA}$-based TRS $\{(7),(9)\}$ cannot be handled by the techniques presented so far. Notice that in any chain, each occurrence of the $\mathcal{PA}$-based rewrite rule (7) is followed by an occurrence of the $\mathcal{PA}$-based rewrite rule (9). Thus, (7) may be replaced by a new $\mathcal{PA}$-based rewrite rule that simulates an application of (7) followed by an application of (9). This new $\mathcal{PA}$-based rewrite rule is

$$\mathsf{eval}_1(x,y,z) \to \mathsf{eval}_1(x-1,y,z)\ [\![x > z \wedge y \not> z]\!] \quad (7.9)$$

The $\mathcal{PA}$-based TRS $\{(7.9),(9)\}$ is first transformed into the $\mathcal{PA}$-based TRS $\{(7.9)\}$ using the termination graph. Then, the $\mathcal{PA}$-based TRS $\{(7.9)\}$ can be handled using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}_1) = x_1 - x_3$. $\Diamond$

Formally, this idea can be stated as the following termination processor. In practice, its main use is the propagation of $\mathcal{PA}$-constraints. Notice that chaining of $\mathcal{PA}$-based rewrite rules is easily possible if the left-hand sides have the form $f(x_1,\ldots,x_n)$. Also, notice that the rule $l \to f(s_1,\ldots,s_n)[\![C]\!]$ is *replaced* by the rules that are obtained by chaining.

**Theorem 26 (Processor Based on Chaining).** *The termination processor with* $\mathsf{Proc}(\mathcal{R} \uplus \{l \to f(s_1,\ldots,s_n)[\![C]\!]\}) = \{\mathcal{R} \cup \mathcal{R}'\}$ *where* $\mathcal{R}' = \{l \to r'\mu[\![C \wedge C'\mu]\!] \mid f(x_1,\ldots,x_n) \to r'[\![C']\!] \in \mathcal{R} \cup \{l \to f(s_1,\ldots,s_n)[\![C]\!]\}, \mu = \{x_1 \mapsto s_1,\ldots,x_n \mapsto s_n\}\}$ *is sound.*

*Proof.* It needs to be shown that every occurrence of (a variable-renamed version of) $l \to r[\![C]\!]$ and the $\mathcal{PA}$-based rewrite rule following it in an infinite chain can be replaced by some $\mathcal{PA}$-based rewrite rule from $\mathcal{R}'$. Thus, assume some infinite chain contains $\ldots, l \to r[\![C]\!], l' \to r'[\![C']\!], v \to w[\![D]\!], \ldots$. Let the infinite chain be based on the substitution $\sigma$, i.e., $r\sigma \simeq_{\mathcal{PA}} l'\sigma$ and $C\sigma$ and $C'\sigma$ are $\mathcal{PA}$-valid. Since $r = l'\mu$, $r\sigma = l'\mu\sigma$ and thus $r\sigma \simeq_{\mathcal{PA}} l'\mu\sigma$. Therefore $l'\mu\sigma \simeq_{\mathcal{PA}} l'\sigma$ and thus $x_i\mu\sigma \simeq_{\mathcal{PA}} x_i\sigma$ for all variables $x$ occurring in $l'$. This implies that $C'\mu\sigma$ is $\mathcal{PA}$-valid since $C'\sigma$ is $\mathcal{PA}$-valid. Thus, $l \to r[\![C]\!], l' \to r'[\![C']\!]$ can be replaced by $l \to r'\mu[\![C \wedge C'\mu]\!]$ since $C\sigma \wedge C'\mu\sigma$ is $\mathcal{PA}$-valid and $r'\mu\sigma \simeq_{\mathcal{PA}} r'\sigma \simeq_{\mathcal{PA}} v\sigma$. $\square$

## 8 Implementation

In order to show the effectiveness and practicality of the proposed approach, it has been implemented in the prototype tool `pasta` ($\mathcal{PA}$-based Term Rewrite System Termination Analyzer). The prototype `pasta` has been written in OCaml and consists of about 1500 lines of code. The input to `pasta` is a $\mathcal{PA}$-based TRS, and it attempts to find a termination proof fully automatically. An empirical evaluation shows that `pasta` is indeed very successful and effective.

The first decision that has to be made is the order in which the termination processors from Sections 5–7 are applied. For this, the loop given in Figure 2 is used. Here, SCC is the termination processor of Theorem 14 that returns the

```
todo := SCC(R)
while todo ≠ ∅ do
  P := pick-and-remove(todo)
  P' := polo(P)
  if P = P' then
    P' := chain(P)
    if P = P' then
      return "Failure"
    end if
  end if
  todo := todo ∪ SCC(P')
end while
return "Termination shown"
```

**Fig. 2.** Main loop of `pasta`.

SCCs of the termination graph, polo is the termination processor of Theorem 17 using linear $\mathcal{PA}$-polynomial interpretations that removes $\mathcal{PA}$-based rewrite rules which are decreasing w.r.t. $\succ_{\mathcal{P}ol}$, and chain is the termination processor of Theorem 26 that combines $\mathcal{PA}$-based rewrite rules.

SCC builds the termination graph using a decision procedure for $\mathcal{PA}$-satisfiability. Then, the standard graph algorithm is used to compute the SCCs. In `pasta`, the library ocamlgraph[5] is used for graph manipulations, and the SMT solver `yices`[6] is used as a decision procedure for $\mathcal{PA}$-satisfiability. The most non-trivial part of the implementation is the function polo for the automatic generation of (linear) $\mathcal{PA}$-polynomial interpretations.

### 8.1 Automatic Generation of $\mathcal{PA}$-Polynomial Interpretations

For the automatic generation, a linear *parametric* $\mathcal{PA}$-polynomial interpretation is used, i.e., an interpretation where the coefficients of the polynomials are

[5] Freely available from `http://ocamlgraph.lri.fr/`
[6] Available from `http://yices.csl.sri.com/`

not integers but parameters that have to be determined. Thus, $\mathcal{P}ol(\mathsf{eval}_i) = a_{i,1}x_1 + \ldots + a_{i,n}x_n + c_i$ for each function symbol $\mathsf{eval}_i$, where the $a_{i,j}$ and $c_i$ are parameters.

In this section, it is assumed that the constraints of all $\mathcal{PA}$-based rewrite rules are conjunctions of atomic $\mathcal{PA}$-constraints. This can be achieved by a conversion into disjunctive normal form (DNF) and the introduction of one rewrite rule for each dual clause in this DNF. Recall that the termination processor of Theorem 17 operating on a $\mathcal{PA}$-based TRS $\mathcal{R}$ aims at generating a $\mathcal{PA}$-polynomial interpretation $\mathcal{P}ol$ with

- $l[\![C]\!] \succ_{\mathcal{P}ol} r[\![C]\!]$ for all $l \to r[\![C]\!] \in \mathcal{R}'$ for some non-empty $\mathcal{R}' \subseteq \mathcal{R}$ and
- $l[\![C]\!] \gtrsim_{\mathcal{P}ol} r[\![C]\!]$ for all $l \to r[\![C]\!] \in \mathcal{R} - \mathcal{R}'$.

As shown in Section 6, it suffices to show that

- $\forall^*.\ C \Rightarrow [l]_{\mathcal{P}ol} - [r]_{\mathcal{P}ol} > 0$ and $\forall^*.\ C \Rightarrow [l]_{\mathcal{P}ol} \geq 0$ are true in the integers for all $l \to r[\![C]\!] \in \mathcal{R}'$ for some non-empty $\mathcal{R}' \subseteq \mathcal{R}$ and
- $\forall^*.\ C \Rightarrow [l]_{\mathcal{P}ol} - [r]_{\mathcal{P}ol} \geq 0$ is true in the integers for all $l \to r[\![C]\!] \in \mathcal{R} - \mathcal{R}'$

where $\forall^*$ denotes the universal closure. Notice that $[l]_{\mathcal{P}ol}$ and $[l]_{\mathcal{P}ol} - [r]_{\mathcal{P}ol}$ are linear parametric polynomials, i.e., polynomials over the variables whose coefficients are linear polynomials over the parameters. For instance, if $[l] = \mathsf{eval}(x, x)$ and $\mathcal{P}ol(\mathsf{eval}) = ax_1 + bx_2 + c$, then $[l]_{\mathcal{P}ol} = (a + b)x + c$.

In order to determine the parameters such that $\forall^*.\ C \Rightarrow [l]_{\mathcal{P}ol} - [r]_{\mathcal{P}ol} \geq 0$ is true in the integers for all $l \to r[\![C]\!] \in \mathcal{R}$, sufficient conditions on the parameters are derived and it is checked whether these conditions are satisfiable. The derivation of the conditions is done independently for the $\mathcal{PA}$-based rewrite rules, but the check for satisfiability of the conditions considers all $\mathcal{PA}$-based rewrite rules since they need to be oriented using the same $\mathcal{PA}$-polynomial interpretation.

For a single $\mathcal{PA}$-based rewrite rule $l \to r[\![C]\!]$, the conditions on the parameters are obtained as follows, where $p = [l]_{\mathcal{P}ol} - [r]_{\mathcal{P}ol}$:

1. $C$ is transformed into a conjunction of atomic $\mathcal{PA}$-constraints of the form $\sum_{i=1}^{n} a_i x_i + c \geq 0$ where $a_1, \ldots, a_n, c \in \mathbb{Z}$.
2. Use the $\mathcal{PA}$-constraints from step 1. to derive upper and/or lower bounds on the variables in $p$.
3. Use the bounds from step 2. to derive conditions on the parameters.

*Step 1: Transformation of $C$.* This is straightforward: $s \simeq t$ is transformed into $s - t \geq 0$ and $t - s \geq 0$, $s \geq t$ is transformed into $s - t \geq 0$, and $s > t$ is transformed into $s - t - 1 \geq 0$.

*Step 2: Deriving upper and/or lower bounds.* The $\mathcal{PA}$-constraints obtained after step 1. might already contain upper and/or lower bounds on the variables, where a lower bound has the form $x+c \geq 0$ and an upper bound has the from $-x+c \geq 0$ for some $c \in \mathbb{Z}$. Otherwise, it might be possible to obtain such bounds as follows.

An atomic constraint of the form $ax+c \geq 0$ with $a \neq 1, -1$ that contains only one variable gives a bound on that variable that can be obtained by dividing by

$|a|$ and rounding. For example, the $\mathcal{PA}$-constraint $2x+3 \geq 0$ is transformed into $x+1 \geq 0$, and $-3x-2 \geq 0$ is transformed into $-x-1 \geq 0$.

An atomic $\mathcal{PA}$-constraint with more than one variable can be used to express a variable $x$ occurring with coefficient 1 in terms of the other variables and a fresh slack variable $w$ with $w \geq 0$. This allows to eliminate $x$ from the polynomial $p$ and at the same time gives the lower bound 0 on the slack variable $w$. For example, $x-2y \geq 0$ can be used to eliminate the variable $x$ by replacing it with $2y+w$. Similar reasoning applies if the variable $x$ occurs with coefficient $-1$.

These ideas are formalized in the transformation rules from Figure 3 that operate on triples $\langle C_1, C_2, q\rangle$ where $C_1$ and $C_2$ are sets of atomic $\mathcal{PA}$-constraints and $q$ is a linear parametric polynomial. Here, $C_1$ only contains $\mathcal{PA}$-constraints of the form $\pm x_i + c \geq 0$ giving upper and/or lower bounds on the variable $x_i$ and $C_2$ contains arbitrary atomic $\mathcal{PA}$-constraints. The initial triple is $\langle \emptyset, C, p\rangle$.

$$\mathsf{Strengthen}\ \frac{C_1,\ C_2 \uplus \{a_i x_i + c \geq 0\},\ q}{C_1 \cup \left\{\frac{a_i}{|a_i|}x_i + \lfloor \frac{c}{|a_i|} \rfloor \geq 0\right\},\ C_2,\ q} \quad \text{if } a_i \neq 0$$

$$\mathsf{Express}^{+}\ \frac{C_1,\ C_2 \uplus \left\{\sum_{i=1}^{n} a_i x_i + c \geq 0\right\},\ q}{C_1 \cup \{w \geq 0\},\ C_2\sigma,\ q\sigma} \quad \begin{array}{l}\text{if } a_j = 1 \text{ and } \sigma \text{ is the substitution}\\ \{x_j \mapsto -\sum_{i\neq j} a_i x_i - c + w\}\\ \text{for a fresh slack variable } w\end{array}$$

$$\mathsf{Express}^{-}\ \frac{C_1,\ C_2 \uplus \left\{\sum_{i=1}^{n} a_i x_i + c \geq 0\right\},\ q}{C_1 \cup \{w \geq 0\},\ C_2\sigma,\ q\sigma} \quad \begin{array}{l}\text{if } a_j = -1 \text{ and } \sigma \text{ is the substitution}\\ \{x_j \mapsto \sum_{i\neq j} a_i x_i + c - w\}\\ \text{for a fresh slack variable } w\end{array}$$

**Fig. 3.** Transformation rules to derive upper and/or lower bounds.

*Step 3: Deriving conditions on the parameters.* After finishing step 2., a final triple $\langle C_1, C_2, q\rangle$ is obtained. If $C_1$ contains more than one bound on a variable $x_i$, then it suffices to consider the maximal lower bound and the minimal upper bound. The bounds in $C_1$ are used in combination with absolute positiveness [18] in order to obtain conditions on the parameters that make $q = \sum_{i=1}^{n} p_i x_i + p_0$ non-negative for all instantiations satisfying $C_1 \cup C_2$.

If $C_1$ contains a lower bound of the form $x_j + c \geq 0$ for the variable $x_j$, then notice that $q = \sum_{i=1}^{n} p_i x_i + p_0$ can also be written as $q = \sum_{i\neq j} p_i x_i + p_j(x_j + c) + p_0 - p_j c$. Since $x_j + c \geq 0$ is assumed, the absolute positiveness test requires $p_j \geq 0$ as a condition on $p_j$. Similarly, if $-x_j + c \geq 0$ occurs in $C_1$, then $q$ can be written as $q = \sum_{i\neq j} p_i x_i - p_j(-x_j + c) + p_0 + p_j c$ and $-p_j \geq 0$ is obtained as a condition on $p_j$. If $C_1$ does not contain any upper or lower bound on a variable $x_j$, then $p_j = 0$ is obtained by the absolute positiveness test. After all variables of $q$ have been processed in this fashion, it additionally needs to be required that the constant term of the final polynomial is non-negative as well.

For example, if $C_1 = \{x+1 \geq 0,\ -y-1 \geq 0\}$ and $q = (a+b)x+by+c$, then $q$ can also be written as $q = (a+b)(x+1) - b(-y-1) + c - (a+b) - b$ and the absolute positiveness test requires $a+b \geq 0$, $-b \geq 0$, and $c-a-2b \geq 0$.

Summarizing this method, the algorithm from Figure 4 is used in order to obtain conditions $D$ on the parameters. Here, $\mathsf{sign}(C)$ is 1 if $C$ is of the form $x_i + c \geq 0$ and $-1$ if $C$ is of the form $-x_i + c \geq 0$.

$D := \mathsf{true}$
$r := p_0$
**for** $1 \leq i \leq n$ **do**
  take constraint $C$ of the form $\pm x_i + c \geq 0$ from $C_1$
  **if** none such $C$ exists **then**
    $D := D \wedge p_i = 0$
  **else**
    $D := D \wedge \mathsf{sign}(C) \cdot p_i \geq 0$
    $r := r - \mathsf{sign}(C) \cdot c \cdot p_i$
  **end if**
**end for**
$D := D \wedge r \geq 0$

**Fig. 4.** Deriving conditions on the parameters

*Automatically finding strictly decreasing rules.* For the termination processor of Theorem 17, it also has to be ensured that $\mathcal{R}'$ is non-empty, i.e., that at least one $\mathcal{PA}$-based rewrite rule is decreasing w.r.t. $\succ_{\mathcal{P}ol}$. Let $l \to r[\![C]\!]$ be a $\mathcal{PA}$-based rewrite rule that should satisfy $l \succ^{C}_{\mathcal{P}ol} r$. Then, $\forall^*.\ C \Rightarrow [l]_{\mathcal{P}ol} \geq 0$ gives rise to conditions $D_1$ on the parameters as above. The second condition, i.e., $\forall^*.\ C \Rightarrow [l]_{\mathcal{P}ol} - [r]_{\mathcal{P}ol} > 0$, gives rise to conditions $D_2$ just as above, with the only difference that the last line of the algorithm now requires $r > 0$.

Given a set of rules $\{l_1 \to r_1[\![C_1]\!], \ldots, l_n \to r_n[\![C_n]\!]\}$, the final constraint on the parameters is then $\bigwedge_{i=1}^{n} D^i \wedge \bigvee_{i=1}^{n} (D_1^i \wedge D_2^i)$ where the $D^i$ are obtained from $\forall^*.\ C_i \Rightarrow [l_i]_{\mathcal{P}ol} - [r_i]_{\mathcal{P}ol} \geq 0$, the $D_1^i$ are obtained from $\forall^*.\ C_i \Rightarrow [l_i]_{\mathcal{P}ol} \geq 0$, and the $D_2^i$ are obtained from $\forall^*.\ C_i \Rightarrow [l_i]_{\mathcal{P}ol} - [r_i]_{\mathcal{P}ol} > 0$.[7] This constraint can be given to a witness-producing decision procedure for $\mathcal{PA}$-satisfiability in order to obtain values for the parameters. $\mathcal{R}'$ can then be obtained easily.

*Example 27.* The method is illustrated on the termination problem $\{(8)\}$ from Example 25 consisting of the rewrite rule $\mathsf{eval}_2(x,y,z) \to \mathsf{eval}_2(x,y-1,z)\ [\![y > z]\!]$. For this, a parametric $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}_2) = ax_1 + bx_2 + cx_3 + d$ is used, where $a, b, c, d$ are parameters that need to be determined. Thus, the goal is to instantiate the parameters in such a way that

[7] It suffices to only consider the new condition $r > 0$ from the last line of the algorithm.

$\mathsf{eval}_2(x,y,z) \succ_{\mathcal{P}ol}^{[\![y>z]\!]} \mathsf{eval}_2(x,y-1,z)$, i.e., such that

$$\forall x,y,z.\ y > z \Rightarrow [\mathsf{eval}_2(x,y,z)]_{\mathcal{P}ol} \geq 0$$
$$\forall x,y,z.\ y > z \Rightarrow [\mathsf{eval}_2(x,y,z)]_{\mathcal{P}ol} - [\mathsf{eval}_2(x,y-1,z)]_{\mathcal{P}ol} > 0$$

are true in the integers. Notice that $[\mathsf{eval}_2(x,y,z)]_{\mathcal{P}ol} = ax + by + cz + d$ and $[\mathsf{eval}_2(x,y-1,z)]_{\mathcal{P}ol} = ax + by + cz - b + d$. Therefore, $[\mathsf{eval}_2(x,y,z)]_{\mathcal{P}ol} - [\mathsf{eval}_2(x,y-1,z)]_{\mathcal{P}ol} = b$.

For the first formula, the constraint $y > z$ is transformed into $y - z - 1 \geq 0$ in step 1. In step 2., the transformation rules from Figure 3 are applied to the triple $\langle \emptyset, \{y - z - 1 \geq 0\}, ax + by + cz + d\rangle$. A possible transformation sequence is as follows, where the $\mathsf{Express}^+$-step uses $\sigma = \{y \mapsto z + w + 1\}$.

$$\mathsf{Express}^+\ \frac{\emptyset,\ \{y - z - 1 \geq 0\},\ ax + by + cz + d}{\{w \geq 0\},\ \emptyset,\ ax + (b+c)z + bw + b + d}$$

Step 3. gives $a = 0 \wedge b + c = 0 \wedge b \geq 0 \wedge b + d \geq 0$ as conditions on the parameters.

For the second formula from above, $b > 0$ is easily obtained as a condition on the parameters. The final constraint on the parameters is thus $a = 0 \wedge b + c = 0 \wedge b \geq 0 \wedge b + d \geq 0 \wedge b > 0$. This constraint is satisfiable and `yices` returns the witness values $a = 0, b = 1, c = -1, d = 0$, giving rise to the $\mathcal{PA}$-polynomial interpretation $\mathcal{P}ol(\mathsf{eval}_2) = x_2 - x_3$ already considered in Example 25. $\Diamond$

## 9 Conclusions

This paper has presenter a method for showing termination of imperative programs operating on integers that is partially based on ideas from the term rewriting literature. For this, a translation from imperative programs into constrained term rewrite systems operating on integers has been introduced. Then, techniques for showing termination of such $\mathcal{PA}$-based TRSs have been developed.

An implementation of this approach has been evaluated on a collection of 40 examples that were taken from various places, including several recent papers on the termination of imperative programs [2–6, 8, 9, 23, 24]. The collection of examples includes "classical" algorithms such as binary search, bubblesort, heapsort, and the computation of the greatest common divisor. Twelve out of these 40 examples (e.g., the heapsort example from [12]) require simple invariants on the program variables (such as "a variable is always non-negative") or simple reasoning of the kind "if variables do not change between control points, then relations that are true for them at the first control point are still true at the second control point" for a successful termination proof. This kind of information can be obtained automatically using static program analysis tools such as `Interproc`[8]. The translation into $\mathcal{PA}$-based TRSs from Section 3 can immediately use this automatically obtained information by adding it to the constraints of the rewrite

[8] Freely available from `http://pop-art.inrialpes.fr/people/bjeannet/bjeannet-forge/interproc/`

rules that are generated. In Appendices A–C, the programs are annotated by the information obtained from the static program analysis by writing conditions of the form `[` $D$ `]`.

The prototype implementation `pasta` has been able to show termination of all examples fully automatically, on average taking less than 0.05 seconds[9] for each example, with the longest time being a little less than a third of a second. Notice that over 80% of the time is spend with calls to `yices`.

| | Proof time (in s) | Time in `yices` (in s) |
|---|---|---|
| Binary Search | 0.284 | 0.254 |
| Bubblesort | 0.042 | 0.029 |
| Heapsort | 0.318 | 0.281 |
| Greatest Common Divisor | 0.032 | 0.027 |
| Average for all 40 examples | 0.048 | 0.039 |
| Total for all 40 examples | 1.910 | 1.557 |

The prototype implementation `pasta` clearly shows the practicality and effectiveness of the proposed approach on a collection of "typical" examples. Notice that an empirical comparison with the methods of [2–6, 8, 9, 23, 24] is not possible since implementation of those methods are not publicly available. The examples, detailed results, the termination proofs generated by `pasta`, and the tool `pasta` itself are available at `http://www.cs.unm.edu/~spf/pasta/`.

[9] All times were obtained on a 2.2 GHz AMD Athlon$^{TM}$ with 2 GB main memory.

# A Examples

## A.1 Bubblesort

The following imperative program fragment describes the updates to the loop indices in the bubblesort algorithm. The annotations were obtained using the interval domain [11].

```
while (x > 0) {
   [ x > 0 ]
   y := 0;
   while (y < x) {
      [ x > 0 && y >= 0 ]
      y++
   }
   [ x > 0 && y >= 0 ]
   x--
}
```

The program is translated into

$$\mathsf{eval}_1(x,y) \to \mathsf{eval}_2(x,0) \qquad [\![x > 0]\!] \tag{1}$$
$$\mathsf{eval}_2(x,y) \to \mathsf{eval}_2(x,y+1)[\![x > 0 \wedge y \geq 0 \wedge y < x]\!] \tag{2}$$
$$\mathsf{eval}_2(x,y) \to \mathsf{eval}_1(x-1,y)[\![x > 0 \wedge y \geq 0 \wedge y \not< x]\!] \tag{3}$$

The termination problem $\{(1),(2),(3)\}$ is transformed into $\{(1),(2)\}$ using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}_1) = \mathcal{P}ol(\mathsf{eval}_2) = x_1$ since

$$\begin{array}{l}\forall x,y.\ x > 0 \Rightarrow x \geq x\\ \forall x,y.\ x > 0 \wedge y \geq 0 \wedge y < x \Rightarrow x \geq x\\ \forall x,y.\ x > 0 \wedge y \geq 0 \wedge y \not< x \Rightarrow x \geq 0\\ \forall x,y.\ x > 0 \wedge y \geq 0 \wedge y \not< x \Rightarrow x > x - 1\end{array}$$

are true in the integers. This new termination problem is then transformed into the termination problem $\{(2)\}$ since the $\mathcal{PA}$-based rewrite rule (1) is not in the SCC of the termination graph. This final termination problem can be handled by the termination processor of Theorem 17 using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}_2) = x_1 - x_2$ since

$$\begin{array}{l}\forall x,y.\ x > 0 \wedge y \geq 0 \wedge y < x \Rightarrow x - y \geq 0\\ \forall x,y.\ x > 0 \wedge y \geq 0 \wedge y < x \Rightarrow x - y > x - y - 1\end{array}$$

are true in the integers.

## A.2 Binary Search

The following imperative program fragment describes the updates to the upper and lower bound used in binary search. Again, the annotations were obtained using the interval domain [11].

```
assume l >= 0 && u >= 0;
while (l < u) {
   either {
      [ l >= 0 && u >= 0 ]
      l := (l + u + 2) / 2
   } or {
      [ l >= 0 && u >= 0 ]
      u := (l + u) / 2
   }
}
```

The program is translated into

$$\mathsf{eval}(l,u) \to \mathsf{eval}(l',u)[\![l \geq 0 \wedge u \geq 0 \wedge l < u \wedge \mathsf{div}(l+u+2,2,l',z)]\!] \quad (1)$$
$$\mathsf{eval}(l,u) \to \mathsf{eval}(l,u')[\![l \geq 0 \wedge u \geq 0 \wedge l < u \wedge \mathsf{div}(l+u,2,u',z)]\!] \quad (2)$$

The termination problem $\{(1),(2)\}$ can be handled using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}) = x_2 - x_1$ since

$$\forall l,u,l',z.\ l \geq 0 \wedge u \geq 0 \wedge l < u \wedge \mathsf{div}(l+u+2,2,l',z) \Rightarrow u - l \geq 0$$
$$\forall l,u,l',z.\ l \geq 0 \wedge u \geq 0 \wedge l < u \wedge \mathsf{div}(l+u+2,2,l',z) \Rightarrow u - l > u - l'$$
$$\forall l,u,l',z.\ l \geq 0 \wedge u \geq 0 \wedge l < u \wedge \mathsf{div}(l+u,2,u',z) \Rightarrow u - l \geq 0$$
$$\forall l,u,l',z.\ l \geq 0 \wedge u \geq 0 \wedge l < u \wedge \mathsf{div}(l+u,2,u',z) \Rightarrow u - l > u' - l$$

are true in the integers.

### A.3 Heapsort

Figure 5 contains an imperative program fragment that describes the updates to the integers variables used in heapsort as given in [12]. In order to simplify presentation, `either` statements with more than one `or` case are used. The annotations were obtained using the interval domain [11]. The program is translated into

$$\mathsf{eval}_0(i,j,l,r,n) \to \mathsf{eval}_1(i,j,l',n,n) \quad [\![n \geq 2 \wedge \mathsf{div}(n+2,2,l',z)]\!] \quad (1)$$
$$\mathsf{eval}_1(i,j,l,r,n) \to \mathsf{eval}_2(i,j,l-1,r,n) \quad [\![l \geq 2]\!] \quad (2)$$
$$\mathsf{eval}_1(i,j,l,r,n) \to \mathsf{eval}_2(i,j,l,r-1,n) \quad [\![l \not\geq 2]\!] \quad (3)$$
$$\mathsf{eval}_2(i,j,l,r,n) \to \mathsf{eval}_3(l,2\cdot l,l,r,n) \quad [\![r \geq 2]\!] \quad (4)$$
$$\mathsf{eval}_3(i,j,l,r,n) \to \mathsf{eval}_4(i,j,l,r,n) \quad [\![j \leq r \wedge j \leq r-1]\!] \quad (5)$$
$$\mathsf{eval}_3(i,j,l,r,n) \to \mathsf{eval}_4(i,j+1,l,r,n) \quad [\![j \leq r \wedge j \leq r-1]\!] \quad (6)$$
$$\mathsf{eval}_3(i,j,l,r,n) \to \mathsf{eval}_3(j,2\cdot j,l,r,n) \quad [\![j \leq r \wedge j \leq r-1 \wedge j \geq 1]\!] \quad (7)$$
$$\mathsf{eval}_3(i,j,l,r,n) \to \mathsf{eval}_3(j+1,2\cdot j+2,l,r,n) \quad [\![j \leq r \wedge j \leq r-1 \wedge j \geq 1]\!] \quad (8)$$
$$\mathsf{eval}_3(i,j,l,r,n) \to \mathsf{eval}_4(i,j,l,r,n) \quad [\![j \leq r \wedge j \not\leq r-1]\!] \quad (9)$$
$$\mathsf{eval}_3(i,j,l,r,n) \to \mathsf{eval}_3(j,2\cdot j,l,r,n) \quad [\![j \leq r \wedge j \not\leq r-1 \wedge j \geq 1]\!] \quad (10)$$
$$\mathsf{eval}_4(i,j,l,r,n) \to \mathsf{eval}_2(i,j,l-1,r,n) \quad [\![l \geq 2 \wedge l \geq 1 \wedge r \geq 2]\!] \quad (11)$$
$$\mathsf{eval}_4(i,j,l,r,n) \to \mathsf{eval}_2(i,j,l,r-1,n) \quad [\![l \not\geq 2 \wedge l \geq 1 \wedge r \geq 2]\!] \quad (12)$$

```
assume n >= 2;
(l, r) := ((n + 2) / 2, n);
if (l >= 2) {
   l--
} else {
   r--
}
while (r >= 2) {
   (i, j) := (l, 2*l);
   while (j <= r) {
      if (j <= r - 1) {
         either {
            break
         } or {
            j++;
            break
         } or {
            [ j >= 1 ]
            (i, j) := (j, 2*j)
         } or {
            [ j >= 1 ]
            (i, j) := (j + 1, 2*j + 2)
         }
      } else {
         either {
            break
         } or {
            [ j >= 1 ]
            (i, j) := (j, 2*j)
         }
      }
   }
   if (l >= 2) {
      [ l >= 1 && r >= 2 ]
      l--
   } else {
      [ l >= 1 && r >= 2 ]
      r--
   }
}
```

**Fig. 5.** Heapsort as used in Appendix A.3.

The termination problem $\{(1)-(12)\}$ is transformed into the termination problem $\{(4)-(12)\}$ since the $\mathcal{PA}$-based rewrite rules $(1)-(3)$ are not in the SCC of the termination graph. Using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}_2) = \mathcal{P}ol(\mathsf{eval}_3) = \mathcal{P}ol(\mathsf{eval}_4) = x_3 + x_4$ this termination problem is transformed into the termination problem $\{(4)-(10)\}$ since

$$\forall i,j,l,r,n.\ r \geq 2 \Rightarrow l + r \geq l + r$$
$$\forall i,j,l,r,n.\ j \leq r \wedge j \leq r - 1 \Rightarrow l + r \geq l + r$$
$$\forall i,j,l,r,n.\ j \leq r \wedge j \leq r - 1 \wedge j \geq 1 \Rightarrow l + r \geq l + r$$
$$\forall i,j,l,r,n.\ j \leq r \wedge j \not\leq r - 1 \Rightarrow l + r \geq l + r$$
$$\forall i,j,l,r,n.\ j \leq r \wedge j \not\leq r - 1 \wedge j \geq 1 \Rightarrow l + r \geq l + r$$
$$\forall i,j,l,r,n.\ l \geq 2 \wedge l \geq 1 \wedge r \geq 2 \Rightarrow l + r \geq 0$$
$$\forall i,j,l,r,n.\ l \geq 2 \wedge l \geq 1 \wedge r \geq 2 \Rightarrow l + r > l + r - 1$$
$$\forall i,j,l,r,n.\ l \not\geq 2 \wedge l \geq 1 \wedge r \geq 2 \Rightarrow l + r \geq 0$$
$$\forall i,j,l,r,n.\ l \not\geq 2 \wedge l \geq 1 \wedge r \geq 2 \Rightarrow l + r > l + r - 1$$

are true in the integers. The termination problem $\{(4) - (10)\}$ is then transformed into the termination problem $\{(7), (8), (10)\}$ since the remaining $\mathcal{PA}$-based rewrite rules are not in the SCC of the termination graph. This final termination problem can be handled using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}_3) = x_4 - x_2$ since

$$\forall i,j,l,r,n.\ j \leq r \wedge j \leq r - 1 \wedge j \geq 1 \Rightarrow r - j \geq 0$$
$$\forall i,j,l,r,n.\ j \leq r \wedge j \leq r - 1 \wedge j \geq 1 \Rightarrow r - j > r - 2 \cdot j$$
$$\forall i,j,l,r,n.\ j \leq r \wedge j \leq r - 1 \wedge j \geq 1 \Rightarrow r - j > r - 2 \cdot j - 2$$
$$\forall i,j,l,r,n.\ j \leq r \wedge j \not\leq r - 1 \wedge j \geq 1 \Rightarrow r - j \geq 0$$
$$\forall i,j,l,r,n.\ j \leq r \wedge j \not\leq r - 1 \wedge j \geq 1 \Rightarrow r - j > r - 2 \cdot j$$

are true in the integers.

### A.4 Simple Increasing Loop

The imperative program fragment

```
while (x > y) {
   y++
}
```

is translated into

$$\mathsf{eval}(x, y) \rightarrow \mathsf{eval}(x, y + 1)[\![x > y]\!] \tag{1}$$

The termination problem $\{(1)\}$ can be handled using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}) = x_1 - x_2$ since

$$\forall x, y.\ x > y \Rightarrow x - y \geq 0$$
$$\forall x, y.\ x > y \Rightarrow x - y > x - y - 1$$

are true in the integers.

### A.5 Increasing Loop with $\mathcal{PA}$-based Condition

The imperative program fragment

```
while (x >= y + 1) {
   y++
}
```

is translated into

$$\mathsf{eval}(x, y) \to \mathsf{eval}(x, y + 1)[\![x \geq y + 1]\!] \tag{1}$$

The termination problem $\{(1)\}$ can be handled using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}) = x_1 - x_2$ since

$$\forall x, y.\ x \geq y + 1 \Rightarrow x - y \geq 0$$
$$\forall x, y.\ x \geq y + 1 \Rightarrow x - y > x - y - 1$$

are true in the integers.

### A.6 Loop with Two Increasing Variables

The imperative program fragment

```
while (x > y + z) {
   (y, z) := (y + 1, z + 1)
}
```

is translated into

$$\mathsf{eval}(x, y, z) \to \mathsf{eval}(x, y + 1, z + 1)[\![x > y + z]\!] \tag{1}$$

The termination problem $\{(1)\}$) can be handled using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}) = x_1 - x_2 - x_3$ since

$$\forall x, y, z.\ x > y + z \Rightarrow x - y - z \geq 0$$
$$\forall x, y, z.\ x > y + z \Rightarrow x - y - z > x - y - z - 2$$

are true in the integers.

### A.7 Boolean Combination in Conditions

The imperative program fragment

```
while (x > y && x > z) {
   (y, z) := (y + 1, z + 1)
}
```

is translated into

$$\mathsf{eval}(x, y, z) \to \mathsf{eval}(x, y + 1, z + 1)[\![x > y \wedge x > z]\!] \tag{1}$$

The termination problem $\{(1)\}$ can be handled using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}) = 2x_1 - x_2 - x_3$ since

$$\forall x, y, z.\ x > y \wedge x > z \Rightarrow 2x - y - z \geq 0$$
$$\forall x, y, z.\ x > y \wedge x > z \Rightarrow 2x - y - z > 2x - y - z - 2$$

are true in the integers.

### A.8 Increase in All Variables

The imperative program fragment

```
while (x > y) {
   (x, y) := (x + 1, y + 2)
}
```

is translated into

$$\mathsf{eval}(x, y) \to \mathsf{eval}(x + 1, y + 2)[\![x > y]\!] \tag{1}$$

The termination problem $\{(1)\}$ can be handled using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}) = x_1 - x_2$ since

$$\forall x, y.\ x > y \Rightarrow x - y \geq 0$$
$$\forall x, y.\ x > y \Rightarrow x - y > x - y - 1$$

are true in the integers.

### A.9 Increase by Addition

The imperative program fragment (where the annotations were obtained using the interval domain [11])

```
assume y > 0;
while (x >= z) {
   [ y > 0 ]
   z := z + y
}
```

is translated into

$$\mathsf{eval}(x, y, z) \to \mathsf{eval}(x, y, z + y)[\![x \geq z \wedge y > 0]\!] \tag{1}$$

The termination problem $\{(1)\}$ can be handled using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}) = x_1 - x_3$ since

$$\forall x, y, z.\ x \geq z \wedge y > 0 \Rightarrow x - z \geq 0$$
$$\forall x, y, z.\ x \geq z \wedge y > 0 \Rightarrow x - z > x - z - y$$

are true in the integers.

### A.10 Increase in Different Variables

The imperative program fragment

```
while (x <> y) {
   if (x > y) {
      y++
   } else {
      x++
   }
}
```

is translated into

$$\mathsf{eval}(x,y) \to \mathsf{eval}(x,y+1)[\![x \not\approx y \wedge x > y]\!] \quad (1)$$
$$\mathsf{eval}(x,y) \to \mathsf{eval}(x+1,y)[\![x \not\approx y \wedge x \not> y]\!] \quad (2)$$

The termination problem $\{(1),(2)\}$ can be handled using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}) = (x_1 - x_2)^2$ since

$$\begin{array}{l}\forall x,y.\ x \neq y \wedge x > y \Rightarrow (x-y)^2 \geq 0\\ \forall x,y.\ x \neq y \wedge x > y \Rightarrow (x-y)^2 \geq (x-y-1)^2\\ \forall x,y.\ x \neq y \wedge x \not> y \Rightarrow (x-y)^2 \geq 0\\ \forall x,y.\ x \neq y \wedge x \not> y \Rightarrow (x-y)^2 \geq (x-y+1)^2\end{array}$$

are true in the integers.

### A.11 Increase and Decrease in Different Variables

The imperative program

```
while (x > y) {
   if (x > z) {
      either {
         y++
      } or {
         z++
      }
   } else {
      x--
   }
}
```

It is translated into the $\mathcal{PA}$-based TRS

$$\mathsf{eval}_1(x,y,z) \to \mathsf{eval}_2(x,y,z) \quad [\![x > y]\!] \quad (1)$$
$$\mathsf{eval}_2(x,y,z) \to \mathsf{eval}_1(x,y+1,z)[\![x > z]\!] \quad (2)$$
$$\mathsf{eval}_2(x,y,z) \to \mathsf{eval}_1(x,y,z+1)[\![x > z]\!] \quad (3)$$
$$\mathsf{eval}_2(x,y,z) \to \mathsf{eval}_1(x-1,y,z)[\![x \not> z]\!] \quad (4)$$

The termination problem $\{(1)-(4)\}$ is transformed into the termination problem $\{(1),(2),(4)\}$ by a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}_1) = \mathcal{P}ol(\mathsf{eval}_2) = x_1 - x_3$ since

$$\begin{array}{l}\forall x,y,z.\ x > y \Rightarrow x - z \geq x - z\\ \forall x,y,z.\ x > z \Rightarrow x - z \geq x - z\\ \forall x,y,z.\ x > z \Rightarrow x - z \geq 0\\ \forall x,y,z.\ x > z \Rightarrow x - z > x - z - 1\\ \forall x,y,z.\ x \not> z \Rightarrow x - z \geq x - z\end{array}$$

are true in the integers. The termination problem $\{(1), (2), (4)\}$ is then transformed into the termination problem $\{(2), (4)\}$ using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}_1) = x_1 - x_2$ and $\mathcal{P}ol(\mathsf{eval}_2) = x_1 - x_2 - 1$ since

$$\begin{array}{l}
\forall x, y, z.\ x > y \Rightarrow x - y \geq 0 \\
\forall x, y, z.\ x > y \Rightarrow x - y > x - y - 1 \\
\forall x, y, z.\ x > z \Rightarrow x - y \geq x - y - 1 \\
\forall x, y, z.\ x \not> z \Rightarrow x - y \geq x - y - 1
\end{array}$$

are true in the integers. This final termination problem can be handled by the termination graph since it does not contain any SCCs.

# B Examples from the Termination Problem Data Base

Here, imperative programs from the directory `TRS/Beerendonk` in the *Termination Problem Data Base*[10] were translated into $\mathcal{PA}$-based TRSs.

## B.1 Example 1

The imperative program fragment

```
while (x > y) {
   x--
}
```

is translated into

$$\mathsf{eval}(x, y) \to \mathsf{eval}(x - 1, y)[\![x > y]\!] \tag{1}$$

The termination problem $\{(1)\}$ can be handled using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}) = x_1 - x_2$ since

$$\begin{array}{l} \forall x, y.\ x > y \Rightarrow x - y \geq 0 \\ \forall x, y.\ x > y \Rightarrow x - y > x - y - 1 \end{array}$$

are true in the integers.

## B.2 Example 2

The imperative program fragment

```
while (x > y) {
   (x, y) := (x - 1, y + 1)
}
```

is translated into

$$\mathsf{eval}(x, y) \to \mathsf{eval}(x - 1, y + 1)[\![x > y]\!] \tag{1}$$

The termination problem $\{(1)\}$ can be handled using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}) = x_1 - x_2$ since

$$\begin{array}{l} \forall x, y.\ x > y \Rightarrow x - y \geq 0 \\ \forall x, y.\ x > y \Rightarrow x - y > x - y - 2 \end{array}$$

are true in the integers.

[10] Available at `http://www.lri.fr/~marche/tpdb/`

### B.3 Example 3

The imperative program fragment (where the annotations were obtained using the interval domain [11]

```
assume x > 0;
while (x > y) {
   [ x > 0 ]
   y := x + y
}
```

is translated into

$$\mathsf{eval}(x, y) \to \mathsf{eval}(x, x + y)[\![x > 0 \wedge x > y]\!] \tag{1}$$

The termination problem $\{(1)\}$ can be handled using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}) = x_1 - x_2$ since

$$\begin{array}{l} \forall x, y.\ x > 0 \wedge x > y \Rightarrow x - y \geq 0 \\ \forall x, y.\ x > 0 \wedge x > y \Rightarrow x - y > -y \end{array}$$

are true in the integers.

### B.4 Example 4

The imperative program fragment

```
while (x > y) {
   (x, y) := (y, x)
}
```

is translated into

$$\mathsf{eval}(x, y) \to \mathsf{eval}(y, x)[\![x > y]\!] \tag{1}$$

The termination problem $\{(1)\}$ can be handled using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}) = x_1 - x_2$ since

$$\begin{array}{l} \forall x, y.\ x > y \Rightarrow x - y \geq 0 \\ \forall x, y.\ x > y \Rightarrow x - y > y - x \end{array}$$

are true in the integers.

### B.5 Example 5

The imperative program fragment

```
while (x > 0 && 2 | x) {
   x--
}
```

is translated into

$$\mathsf{eval}(x) \to \mathsf{eval}(x-1)[\![x > 0 \wedge 2 \mid x]\!] \tag{1}$$

The termination problem $\{(1)\}$ be handled using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}) = x_1$ since

$$\begin{array}{l} \forall x.\ x > 0 \wedge 2 \mid x \Rightarrow x \geq 0 \\ \forall x.\ x > 0 \wedge 2 \mid x \Rightarrow x > x - 1 \end{array}$$

are true in the integers.

### B.6 Example 6

The imperative program fragment

```
while (x > 0 && y > 0) {
   (x, y) := (x - 1, y - 1)
}
```

is translated into

$$\mathsf{eval}(x, y) \to \mathsf{eval}(x-1, y-1)[\![x > 0 \wedge y > 0]\!] \tag{1}$$

The termination problem $\{(1)\}$ can be handled using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}) = x_1$ since

$$\begin{array}{l} \forall x, y.\ x > 0 \wedge y > 0 \Rightarrow x \geq 0 \\ \forall x, y.\ x > 0 \wedge y > 0 \Rightarrow x \geq x - 1 \end{array}$$

are true in the integers.

### B.7 Example 7

The imperative program fragment

```
while (x > z && y > z) {
   (x, y) := (x - 1, y - 1)
}
```

is translated into

$$\mathsf{eval}(x, y, z) \to \mathsf{eval}(x-1, y-1, z)[\![x > z \wedge y > z]\!] \tag{1}$$

The termination problem $\{(1)\}$ can be handled using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}) = x_1 - x_3$ since

$$\begin{array}{l} \forall x, y, z.\ x > z \wedge y > z \Rightarrow x - z \geq 0 \\ \forall x, y, z.\ x > z \wedge y > z \Rightarrow x - z > x - z - 1 \end{array}$$

are true in the integers.

### B.8 Example 8

The imperative program fragment (where the annotations were obtained using the interval domain [11])

```
assume x > 0;
while (x <> 0) {
   if (2 | x) {
      [ x > 0 ]
      x := x / 2
   } else {
      [ x > 0 ]
      x--
   }
}
```

is translated into

$$\mathsf{eval}(x) \to \mathsf{eval}(x-1)[\![x > 0 \wedge x \not\simeq 0 \wedge 2 \not| x]\!] \quad (1)$$
$$\mathsf{eval}(x) \to \mathsf{eval}(x') \quad [\![x > 0 \wedge x \not\simeq 0 \wedge 2 \mid x \wedge \mathsf{div}(x, 2, x', y)]\!] \quad (2)$$

The termination problem $\{(1), (2)\}$ can be handled using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}) = x_1$ since

$$\forall x.\ x > 0 \wedge x \neq 0 \wedge 2 \not| x \Rightarrow x \geq 0$$
$$\forall x.\ x > 0 \wedge x \neq 0 \wedge 2 \not| x \Rightarrow x > x - 1$$
$$\forall x, x', y.\ x > 0 \wedge x \neq 0 \wedge 2 \mid x \wedge \mathsf{div}(x, 2, x', y) \Rightarrow x \geq 0$$
$$\forall x, x', y.\ x > 0 \wedge x \neq 0 \wedge 2 \mid x \wedge \mathsf{div}(x, 2, x', y) \Rightarrow x > x'$$

are true in the integers.

### B.9 Example 9

The imperative program fragment (where the annotations were obtained using the interval domain [11])

```
assume x > 0;
while (x <> 0) {
   if (x > y) {
      [ x > 0 ]
      x := y
   } else {
      [ x > 0 ]
      x--
   }
}
```

is translated into

$$\mathsf{eval}(x, y) \to \mathsf{eval}(x-1, y)[\![x > 0 \wedge x \not\simeq 0 \wedge x \not> y]\!] \quad (1)$$
$$\mathsf{eval}(x, y) \to \mathsf{eval}(y, y) \quad [\![x > 0 \wedge x \not\simeq 0 \wedge x > y]\!] \quad (2)$$

The termination problem $\{(1), (2)\}$ can be handled using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}) = x_1$ since

$$\begin{array}{l}\forall x, y.\ x > 0 \wedge x \neq 0 \wedge x \not> y \Rightarrow x \geq 0\\ \forall x, y.\ x > 0 \wedge x \neq 0 \wedge x \not> y \Rightarrow x > x - 1\\ \forall x, y.\ x > 0 \wedge x \neq 0 \wedge x > y \Rightarrow x \geq 0\\ \forall x, y.\ x > 0 \wedge x \neq 0 \wedge x > y \Rightarrow x > y\end{array}$$

are true in the integers.

### B.10 Example 10

The imperative program fragment

```
while (x + y > 0) {
   if (x > 0) {
      x--
   } else if (y > 0) {
      y--
   } else {
      skip
   }
}
```

is translated into

$$\mathsf{eval}(x, y) \rightarrow \mathsf{eval}(x - 1, y)[\![x + y > 0 \wedge x > 0]\!] \quad (1)$$
$$\mathsf{eval}(x, y) \rightarrow \mathsf{eval}(x, y - 1)[\![x + y > 0 \wedge x \not> 0 \wedge y > 0]\!] \quad (2)$$
$$\mathsf{eval}(x, y) \rightarrow \mathsf{eval}(x, y) \quad [\![x + y > 0 \wedge x \not> 0 \wedge y \not> 0]\!] \quad (3)$$

The termination problem $\{(1), (2), (3)\}$ is transformed into the termination problem $\{(1), (2)\}$ using the termination graph since the constraint of the $\mathcal{PA}$-based rewrite rule (3) is $\mathcal{PA}$-unsatisfiable. This termination problem can be handled using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}) = x_1 + x_2$ since

$$\begin{array}{l}\forall x, y.\ x + y > 0 \wedge x > 0 \Rightarrow x + y \geq 0\\ \forall x, y.\ x + y > 0 \wedge x > 0 \Rightarrow x + y > x + y - 1\\ \forall x, y.\ x + y > 0 \wedge x \not> 0 \wedge y > 0 \Rightarrow x + y \geq 0\\ \forall x, y.\ x + y > 0 \wedge x \not> 0 \wedge y > 0 \Rightarrow x + y > x + y - 1\end{array}$$

are true in the integers.

### B.11 Example 11

The imperative program fragment

```
while (x + y > 0) {
   if (x > y) {
      x--
   else if (x = y) {
      x--
   } else {
      y--
   }
}
```

is translated into

$$\mathsf{eval}(x, y) \to \mathsf{eval}(x - 1, y)[\![x + y > 0 \wedge x > y]\!] \quad (1)$$
$$\mathsf{eval}(x, y) \to \mathsf{eval}(x - 1, y)[\![x + y > 0 \wedge x \not> y \wedge x \simeq y]\!] \quad (2)$$
$$\mathsf{eval}(x, y) \to \mathsf{eval}(x, y - 1)[\![x + y > 0 \wedge x \not> y \wedge x \not\simeq y]\!] \quad (3)$$

The termination problem $\{(1), (2), (3)\}$ can be handled using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}) = x_1 + x_2$ since

$$\begin{array}{l}
\forall x, y.\ x + y > 0 \wedge x > y \Rightarrow x + y \geq 0 \\
\forall x, y.\ x + y > 0 \wedge x > y \Rightarrow x + y > x + y - 1 \\
\forall x, y.\ x + y > 0 \wedge x \not> y \wedge x = y \Rightarrow x + y \geq 0 \\
\forall x, y.\ x + y > 0 \wedge x \not> y \wedge x = y \Rightarrow x + y > x + y - 1 \\
\forall x, y.\ x + y > 0 \wedge x \not> y \wedge x \neq y \Rightarrow x + y \geq 0 \\
\forall x, y.\ x + y > 0 \wedge x \not> y \wedge x \neq y \Rightarrow x + y > x + y - 1
\end{array}$$

are true in the integers.

### B.12 Example 12

The imperative program fragment

```
while (x > 0 || y > 0) {
   if (x > 0) {
      x--
   } else if (y > 0) {
      y--
   } else {
      skip
   }
}
```

is translated into

$$\mathsf{eval}(x, y) \to \mathsf{eval}(x - 1, y)[\![(x > 0 \vee y > 0) \wedge x > 0]\!] \quad (1)$$
$$\mathsf{eval}(x, y) \to \mathsf{eval}(x, y - 1)[\![(x > 0 \vee y > 0) \wedge x \not> 0 \wedge y > 0]\!] \quad (2)$$
$$\mathsf{eval}(x, y) \to \mathsf{eval}(x, y) \quad [\![(x > 0 \vee y > 0) \wedge x \not> 0 \wedge y \not> 0]\!] \quad (3)$$

The termination problem $\{(1),(2),(3)\}$ is transformed into the termination problem $\{(1),(2)\}$ using the termination graph since the constraints of the $\mathcal{PA}$-based rewrite rule (3) is $\mathcal{PA}$-unsatisfiable. This termination problem can be transformed into the termination problem $\{(2)\}$ using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}) = x_1$ since

$$\begin{array}{l}\forall x, y.\ (x > 0 \vee y > 0) \wedge x > 0 \Rightarrow x \geq 0\\ \forall x, y.\ (x > 0 \vee y > 0) \wedge x > 0 \Rightarrow x > x - 1\\ \forall x, y.\ (x > 0 \vee y > 0) \wedge x \not> 0 \wedge y > 0 \Rightarrow x \geq x\end{array}$$

are true in the integers. The termination problem $\{(2)\}$ can now be handled using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}) = x_2$ since

$$\begin{array}{l}\forall x, y.\ (x > 0 \vee y > 0) \wedge x \not> 0 \wedge y > 0 \Rightarrow y \geq 0\\ \forall x, y.\ (x > 0 \vee y > 0) \wedge x \not> 0 \wedge y > 0 \Rightarrow y > y - 1\end{array}$$

are true in the integers.

### B.13 Example 13

The imperative program fragment

```
while (x > z || y > z) {
   if (x > z) {
      x--
   } else if (y > z) {
      y--
   } else {
      skip
   }
}
```

is translated into

$$\mathsf{eval}(x, y, z) \rightarrow \mathsf{eval}(x - 1, y, z)[\![(x > z \vee y > z) \wedge x > z]\!] \quad (1)$$

$$\mathsf{eval}(x, y, z) \rightarrow \mathsf{eval}(x, y - 1, z)[\![(x > z \vee y > z) \wedge x \not> z \wedge y > z]\!] \quad (2)$$

$$\mathsf{eval}(x, y, z) \rightarrow \mathsf{eval}(x, y, z) \quad [\![(x > z \vee y > z) \wedge x \not> z \wedge y \not> z]\!] \quad (3)$$

The termination problem $\{(1),(2),(3)\}$ is transformed into the termination problem $\{(1),(2)\}$ using the termination graph since the constraints of the $\mathcal{PA}$-based rewrite rule (3) is $\mathcal{PA}$-unsatisfiable. This termination problem can be transformed into the termination problem $\{(2)\}$ using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}) = x_1 - x_3$ since

$$\begin{array}{l}\forall x, y, z.\ (x > z \vee y > z) \wedge x > z \Rightarrow x - z \geq 0\\ \forall x, y, z.\ (x > z \vee y > z) \wedge x > z \Rightarrow x - z > x - z - 1\\ \forall x, y, z.\ (x > z \vee y > z) \wedge x \not> z \wedge y > z \Rightarrow x - z \geq x - z\end{array}$$

are true in the integers. The termination problem $\{(2)\}$ can now be handled using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}) = x_2 - x_3$ since

$$\forall x, y, z.\ (x > z \vee y > z) \wedge x \not> z \wedge y > z \Rightarrow y - z \geq 0$$
$$\forall x, y, z.\ (x > z \vee y > z) \wedge x \not> z \wedge y > z \Rightarrow y - z > y - z - 1$$

are true in the integers.

### B.14 Example 14

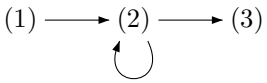
The imperative program fragment

```
while (x = y && x > 0) {
   while (y > 0) {
      (x, y) := (x - 1, y - 1)
   }
}
```

is translated into

$$\mathsf{eval}_1(x, y) \to \mathsf{eval}_2(x, y) \qquad [\![x \simeq y \wedge x > 0]\!] \tag{1}$$
$$\mathsf{eval}_2(x, y) \to \mathsf{eval}_2(x - 1, y - 1) [\![y > 0]\!] \tag{2}$$
$$\mathsf{eval}_2(x, y) \to \mathsf{eval}_1(x, y) \qquad [\![y \not> 0]\!] \tag{3}$$

Using the termination graph




the termination processor of Theorem 14 produces the termination problem $\{(2)\}$. This termination problem can be handled using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}_2) = x_2$ since

$$\forall x, y.\ y > 0 \Rightarrow y \geq 0$$
$$\forall x, y.\ y > 0 \Rightarrow y > y - 1$$

are true in the integers.

### B.15 Example 15

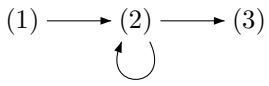
The imperative program fragment

```
while (x = y && x > z) {
   while (y > z) {
      (x, y) := (x - 1, y - 1)
   }
}
```

is translated into

$$\mathsf{eval}_1(x, y, z) \to \mathsf{eval}_2(x, y, z) \qquad [\![x \simeq y \wedge x > z]\!] \tag{1}$$
$$\mathsf{eval}_2(x, y, z) \to \mathsf{eval}_2(x - 1, y - 1, z) [\![y > z]\!] \tag{2}$$
$$\mathsf{eval}_2(x, y, z) \to \mathsf{eval}_1(x, y, z) \qquad [\![y \not> z]\!] \tag{3}$$

Using the termination graph




the termination processor of Theorem 14 produces the termination problem $\{(2)\}$. This termination problem can be handled using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}_2) = x_2 - x_3$ since

$$\forall x, y.\ y > z \Rightarrow y - z \geq 0$$
$$\forall x, y.\ y > z \Rightarrow y - z > y - z - 1$$

are true in the integers.

### B.16 Example 16

The imperative program fragment (where the annotations were obtained using the interval domain [11])

```
while (x > 0) {
   while (y > 0) {
      [ x > 0 ]
      y--
   }
   [ x > 0 ]
   x--
}
```

is translated into

$$\mathsf{eval}_1(x, y) \rightarrow \mathsf{eval}_2(x, y) \quad [\![x > 0]\!] \tag{1}$$
$$\mathsf{eval}_2(x, y) \rightarrow \mathsf{eval}_2(x, y - 1)[\![x > 0 \wedge y > 0]\!] \tag{2}$$
$$\mathsf{eval}_2(x, y) \rightarrow \mathsf{eval}_1(x - 1, y)[\![x > 0 \wedge y \not> 0]\!] \tag{3}$$

The termination problem $\{(1), (2), (3)\}$ is transformed into $\{(1), (2)\}$ using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}_1) = \mathcal{P}ol(\mathsf{eval}_2) = x_1$ since

$$\forall x, y.\ x > 0 \Rightarrow x \geq x$$
$$\forall x, y.\ x > 0 \wedge y > 0 \Rightarrow x \geq x$$
$$\forall x, y.\ x > 0 \wedge y \not> 0 \Rightarrow x \geq 0$$
$$\forall x, y.\ x > 0 \wedge y \not> 0 \Rightarrow x > x - 1$$

are true in the integers. The termination problem $\{(1), (2)\}$ is transformed into the termination problem $\{(2)\}$ since the $\mathcal{PA}$-based rewrite rule (1) is not in the SCC of the problem's termination graph. Finally, the termination problem $\{(2)\}$ can be handled using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}_2) = x_2$ since

$$\forall x, y.\ x > 0 \wedge y > 0 \Rightarrow y \geq 0$$
$$\forall x, y.\ x > 0 \wedge y > 0 \Rightarrow y > y - 1$$

are true in the integers.

### B.17 Example 17

The imperative program fragment (where the annotations were obtained using the octagon domain [21])

```
while (x > z) {
   while (y > z) {
      [ x > z ]
      y--
   }
   [ x > z ]
   x--
}
```

is translated into

$$\mathsf{eval}_1(x, y, z) \to \mathsf{eval}_2(x, y, z) \quad [\![x > z]\!] \tag{1}$$
$$\mathsf{eval}_2(x, y, z) \to \mathsf{eval}_2(x, y - 1, z)[\![x > z \wedge y > z]\!] \tag{2}$$
$$\mathsf{eval}_2(x, y, z) \to \mathsf{eval}_1(x - 1, y, z)[\![x > z \wedge y \not> z]\!] \tag{3}$$

The termination problem $\{(1), (2), (3)\}$ is transformed into $\{(1), (2)\}$ using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}_1) = \mathcal{P}ol(\mathsf{eval}_2) = x_1 - x_3$ since

$$\begin{array}{l}
\forall x, y, z.\ x > z \Rightarrow x - z \geq x - z \\
\forall x, y, z.\ x > z \wedge y > z \Rightarrow x - z \geq x - z \\
\forall x, y, z.\ x > z \wedge y \not> z \Rightarrow x - z \geq 0 \\
\forall x, y, z.\ x > z \wedge y \not> z \Rightarrow x - z > x - z - 1
\end{array}$$

are true in the integers. The termination problem $\{(1), (2)\}$ is transformed into the termination problem $\{(2)\}$ since the $\mathcal{PA}$-based rewrite rule (1) is not in the SCC of the problem's termination graph. Finally, the termination problem $\{(2)\}$ can be handled using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}_2) = x_2 - x_3$ since

$$\begin{array}{l}
\forall x, y, z.\ x > z \wedge y > z \Rightarrow y - z \geq 0 \\
\forall x, y, z.\ x > z \wedge y > z \Rightarrow y - z > y - z - 1
\end{array}$$

are true in the integers.

### B.18 Example 18

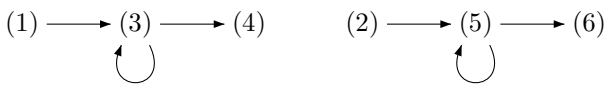
The imperative program fragment

```
while (x > 0 && y > 0) {
   if (x > y) {
      while (x > 0) {
        x--
      }
   } else {
      while (y > 0) {
         y--
      }
   }
}
```

is translated into

$$\mathsf{eval}_1(x,y) \to \mathsf{eval}_2(x,y) \quad [\![x > 0 \wedge y > 0 \wedge x > y]\!] \tag{1}$$
$$\mathsf{eval}_1(x,y) \to \mathsf{eval}_3(x,y) \quad [\![x > 0 \wedge y > 0 \wedge x \not> y]\!] \tag{2}$$
$$\mathsf{eval}_2(x,y) \to \mathsf{eval}_2(x-1,y) [\![x > 0]\!] \tag{3}$$
$$\mathsf{eval}_2(x,y) \to \mathsf{eval}_1(x,y) \quad [\![x \not> 0]\!] \tag{4}$$
$$\mathsf{eval}_3(x,y) \to \mathsf{eval}_3(x,y-1) [\![y > 0]\!] \tag{5}$$
$$\mathsf{eval}_3(x,y) \to \mathsf{eval}_1(x,y) \quad [\![y \not> 0]\!] \tag{6}$$

Using the termination graph




the termination processor of Theorem 14 produces the termination problems $\{(3)\}$ and $\{(4)\}$. The first termination problem can be handled using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}_2) = x_1$ since

$$\forall x, y.\ x > 0 \Rightarrow x \geq 0$$
$$\forall x, y.\ x > 0 \Rightarrow x > x - 1$$

are true in the integers. The second termination problem can be handled similarly using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}_3) = x_2$ since

$$\forall x, y.\ y > 0 \Rightarrow y \geq 0$$
$$\forall x, y.\ y > 0 \Rightarrow y > y - 1$$

are true in the integers.

# C Examples from the Literature

## C.1 Example of Podelski and Rybalchenko

This example is the running example from [24]. The annotations were obtained using the interval domain [11]. The imperative program fragment

```
while (x >= 0) {
   [ x >= 0 ]
   y := 1;
   while (x > y) {
      [ x >= 0 && y > 0 ]
      y := 2 * y
   }
   [ x >= 0 && y > 0 ]
   x--
}
```

is translated into

$$\mathsf{eval}_1(x,y) \to \mathsf{eval}_2(x,1) \quad [\![x \geq 0]\!] \tag{1}$$
$$\mathsf{eval}_2(x,y) \to \mathsf{eval}_2(x,2y) \quad [\![x \geq 0 \wedge y > 0 \wedge x > y]\!] \tag{2}$$
$$\mathsf{eval}_2(x,y) \to \mathsf{eval}_1(x-1,y) [\![x \geq 0 \wedge y > 0 \wedge x \not> y]\!] \tag{3}$$

The termination problem $\{(1),(2),(3)\}$) is transformed into the termination problem $\{(1),(2)\}$ by using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}_1) = \mathcal{P}ol(\mathsf{eval}_2) = x_1$ since

$$\begin{array}{l}\forall x,y.\ x \geq 0 \Rightarrow x \geq x\\ \forall x,y.\ x \geq 0 \wedge y > 0 \wedge x > y \Rightarrow x \geq x\\ \forall x,y.\ x \geq 0 \wedge y > 0 \wedge x \not> y \Rightarrow x \geq 0\\ \forall x,y.\ x \geq 0 \wedge y > 0 \wedge x \not> y \Rightarrow x > x - 1\end{array}$$

are true in the integers. The termination problem $\{(1),(2)\}$ is transformed into $\{(2)\}$ since (1) is not in the SCC of the problem's termination graph. This final termination problem can be handled using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}_2) = x_1 - x_2$ since

$$\begin{array}{l}\forall x,y.\ x \geq 0 \wedge y > 0 \wedge x > y \Rightarrow x - y \geq 0\\ \forall x,y.\ x \geq 0 \wedge y > 0 \wedge x > y \Rightarrow x - y > x - 2y\end{array}$$

are true in the integers.

## C.2 First Example of Cook *et al.*

This example is the (only) example from [8]. The annotations were obtained using the interval domain [11]. The imperative program fragment

```
while (x >= 0) {
   [ x >= 0 ]
   (x, y) := (x + 1, 1);
   while (x >= y) {
      [ x >= 0 && y > 0 ]
      y++
   }
   [ x >= 0 && y > 0 ]
   x := x - 2
}
```

is translated into

$$\mathsf{eval}_1(x, y) \to \mathsf{eval}_2(x + 1, 1)[\![x \geq 0]\!] \tag{1}$$
$$\mathsf{eval}_2(x, y) \to \mathsf{eval}_2(x, y + 1)[\![x \geq 0 \wedge y > 0 \wedge x \geq y]\!] \tag{2}$$
$$\mathsf{eval}_2(x, y) \to \mathsf{eval}_1(x - 2, y)[\![x \geq 0 \wedge y > 0 \wedge x \not\geq y]\!] \tag{3}$$

The termination problem $\{(1), (2), (3)\}$) is transformed into $\{(1), (2)\}$ using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}_1) = x_1 + 1$ and $\mathcal{P}ol(\mathsf{eval}_2) = x_1$ since

$$\begin{array}{l}
\forall x, y.\ x \geq 0 \Rightarrow x + 1 \geq x + 1 \\
\forall x, y.\ x \geq 0 \wedge y > 0 \wedge x \geq y \Rightarrow x \geq x \\
\forall x, y.\ x \geq 0 \wedge y > 0 \wedge x \not\geq y \Rightarrow x \geq 0 \\
\forall x, y.\ x \geq 0 \wedge y > 0 \wedge x \not\geq y \Rightarrow x > x - 1
\end{array}$$

are true in the integers. The termination problem $\{(1), (2)\}$ is transformed into $\{(2)\}$ since (1) is not in the SCC of the problem's termination graph. This final termination problem can be handled using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}_2) = x_1 - x_2$ since

$$\begin{array}{l}
\forall x, y.\ x \geq 0 \wedge y > 0 \wedge x \geq y \Rightarrow x - y \geq 0 \\
\forall x, y.\ x \geq 0 \wedge y > 0 \wedge x \not\geq y \Rightarrow x - y > x - y - 1
\end{array}$$

are true in the integers.

### C.3 Second Example of Cook *et al.*

This example is taken from [9, Figure 3]. The imperative program fragment

```
while (x < y) {
   if (x < z) {
      x++
   } else {
      z++
   }
}
```

is translated into

$$\mathsf{eval}(x,y,z) \to \mathsf{eval}(x+1,y,z)[\![x<y \wedge x<z]\!] \quad (1)$$
$$\mathsf{eval}(x,y,z) \to \mathsf{eval}(x,y,z+1)[\![x<y \wedge x \not< z]\!] \quad (2)$$

The termination problem $\{(1),(2)\}$ is transformed into $\{(2)\}$ with the help of a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}) = x_2 - x_1$ since

$$\forall x,y,z.\ x<y \wedge x<z \Rightarrow y-x \geq 0$$
$$\forall x,y,z.\ x<y \wedge x<z \Rightarrow y-x > y-x-1$$
$$\forall x,y,z.\ x<y \wedge x \not< z \Rightarrow y-x \geq y-x$$

are true in the integers. The termination problem $\{(2)\}$ can be handled using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}) = x_1 - x_3$ since

$$\forall x,y,z.\ x<y \wedge x \not< z \Rightarrow x-z \geq 0$$
$$\forall x,y,z.\ x<y \wedge x \not< z \Rightarrow x-z > x-z-1$$

are true in the integers.

### C.4 Third Example of Cook *et al.*

This example is taken from [9, Figure 11]. The annotations were obtained using the interval domain [11]. The imperative program fragment

```
if (y > 0) {
   while (x < y && y < z) {
      either {
         [ y > 0 ]
         x = x + y
      } or {
         [ y > 0 ]
         z = x - y
      }
   }
}
```

is translated into

$$\mathsf{eval}_0(x,y,z) \to \mathsf{eval}_1(x,y,z) \qquad [\![y>0]\!] \quad (1)$$
$$\mathsf{eval}_1(x,y,z) \to \mathsf{eval}_1(x+y,y,z)[\![x<y \wedge y<z \wedge y>0]\!] \quad (2)$$
$$\mathsf{eval}_1(x,y,z) \to \mathsf{eval}_1(x,y,x-y)[\![x<y \wedge y<z \wedge y>0]\!] \quad (3)$$

The termination problem $\{(1),(2),(3)\}$ is transformed into $\{(2),(3)\}$ since the $\mathcal{PA}$-based rewrite rule (1) is not in the SCC of the termination graph. This new termination problem is transformed into the termination problem $\{(3)\}$ with the help of a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}_1) = x_2 - x_1$ since

$$\forall x, y, z.\ x < y \wedge y < z \wedge y > 0 \Rightarrow y - x \geq 0$$
$$\forall x, y, z.\ x < y \wedge y < z \wedge y > 0 \Rightarrow y - x > -x$$
$$\forall x, y, z.\ x < y \wedge y < z \wedge y > 0 \Rightarrow y - x \geq y - x$$

are true in the integers. The termination problem $\{(3)\}$ can be handled using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}_1) = x_3$ since

$$\forall x, y, z.\ x < y \wedge y < z \wedge y > 0 \Rightarrow z \geq 0$$
$$\forall x, y, z.\ x < y \wedge y < z \wedge y > 0 \Rightarrow z > x - y$$

are true in the integers.

### C.5 First Example of Bradley *et al.*

This example is a minor variation of the main example from [2]. The program computes the greatest common divisor of two positive integers. The annotations were obtained using the polyhedra domain [12]. The imperative program fragment

```
assume x > 0 && y > 0;
while (x <> y) {
   if (x > y) {
      [ x > 0 && y > 0 ]
      x := x - y
   } else {
      [ x > 0 && y > 0 ]
      y := y - x
   }
}
```

is translated into

$$\mathsf{eval}(x, y) \rightarrow \mathsf{eval}(x - y, y)[\![x \not\simeq y \wedge x > 0 \wedge y > 0 \wedge x > y]\!] \quad (1)$$
$$\mathsf{eval}(x, y) \rightarrow \mathsf{eval}(x, y - x)[\![x \not\simeq y \wedge x > 0 \wedge y > 0 \wedge x \not> y]\!] \quad (2)$$

The termination problem $\{(1), (2)\}$ can be handled using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}) = x_1 + x_2$ since

$$\forall x, y.\ x \neq y \wedge x > 0 \wedge y > 0 \wedge x > y \Rightarrow x + y \geq 0$$
$$\forall x, y.\ x \neq y \wedge x > 0 \wedge y > 0 \wedge x > y \Rightarrow x + y > x$$
$$\forall x, y.\ x \neq y \wedge x > 0 \wedge y > 0 \wedge x \not> y \Rightarrow x + y \geq 0$$
$$\forall x, y.\ x \neq y \wedge x > 0 \wedge y > 0 \wedge x \not> y \Rightarrow x + y > y$$

are true in the integers.

### C.6 Second Example of Bradley *et al.*

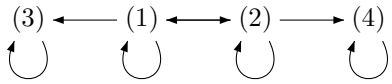
This example stems from [3]. The imperative program fragment

```
while (i < an || j < bn) {
   if (i >= an) {
      cn++;
      j++;
   } else if (j >= bn) {
      cn++;
      i++;
   } else {
      either {
         cn++;
         i++;
      } or {
         cn++;
         j++;
      }
   }
}
```

is translated into

$$\mathsf{eval}(an, bn, cn, i, j) \to \mathsf{eval}(an, bn, cn + 1, i, j + 1)[\![bn > j \wedge an > i]\!] \quad (1)$$
$$\mathsf{eval}(an, bn, cn, i, j) \to \mathsf{eval}(an, bn, cn + 1, i + 1, j)[\![bn > j \wedge an > i]\!] \quad (2)$$
$$\mathsf{eval}(an, bn, cn, i, j) \to \mathsf{eval}(an, bn, cn + 1, i + 1, j)[\![j \geq bn \wedge an > i]\!] \quad (3)$$
$$\mathsf{eval}(an, bn, cn, i, j) \to \mathsf{eval}(an, bn, cn + 1, i, j + 1)[\![bn > j \wedge i \geq an]\!] \quad (4)$$

Using the termination graph




the termination processor of Theorem 14 produces the termination problems $\{(1), (2)\}$, $\{(3)\}$, and $\{(4)\}$.

The termination problem $\{(1), (2)\}$ can be handled using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}) = x_1 + x_2 - x_4 - x_5$ since

$$\forall an, bn, cn, i, j.\ bn > j \wedge an > i \Rightarrow an + bn - i - j \geq 0$$
$$\forall an, bn, cn, i, j.\ bn > j \wedge an > i \Rightarrow an + bn - i - j > an + bn - i - (j + 1)$$
$$\forall an, bn, cn, i, j.\ bn > j \wedge an > i \Rightarrow an + bn - i - j > an + bn - (i + 1) - j$$

are true in the integers.

The termination problem $\{(3)\}$ can be handled using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}) = x_1 - x_4$ since

$$\forall an, bn, cn, i, j.\ j \geq bn \wedge an > i \Rightarrow an - i \geq 0$$
$$\forall an, bn, cn, i, j.\ j \geq bn \wedge an > i \Rightarrow an - i > an - (i + 1)$$

are true in the integers.

The termination problem $\{(4)\}$ can be handled using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}) = x_2 - x_5$ since

$$\forall an, bn, cn, i, j.\ bn > j \wedge i \geq an \Rightarrow bn - j \geq 0$$
$$\forall an, bn, cn, i, j.\ bn > j \wedge i \geq an \Rightarrow bn - j > bn - (j + 1)$$

are true in the integers.

### C.7 First Example of Colon and Sipma

This example stems from [5]. The imperative program fragment

```
while (i <= 100 && j <= k) {
   (i, j) := (j, i + 1);
   k--;
}
```

is translated into

$$\mathsf{eval}(i, j, k) \rightarrow \mathsf{eval}(j, i + 1, k)[\![i \leq 100 \wedge j \leq k]\!] \tag{1}$$

The termination problem $\{(1)\}$ can be handled using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}) = -x_1 - x_2 + x_3 + 100$ since

$$\forall i, j, k.\ i \leq 100 \wedge j \leq k \Rightarrow -i - j + k + 100 \geq 0$$
$$\forall i, j, k.\ i \leq 100 \wedge j \leq k \Rightarrow -i - j + k + 100 > -j - (i + 1) + k + 100$$

are true in the integers.

### C.8 Second Example of Colon and Sipma

This example stems from [6]. The imperative program fragment

```
while (i >= 0) {
   j := 0;
   while (j <= i - 1) {
      j++;
   }
   i--;
}
```

is translated into

$$\mathsf{eval}_1(i, j) \rightarrow \mathsf{eval}_2(i, 0) \qquad [\![i \geq 0]\!] \tag{1}$$
$$\mathsf{eval}_2(i, j) \rightarrow \mathsf{eval}_2(i, j + 1)[\![j \leq i - 1]\!] \tag{2}$$
$$\mathsf{eval}_2(i, j) \rightarrow \mathsf{eval}_1(i - 1, j)[\![j > i - 1]\!] \tag{3}$$

The termination problem $\{(1), (2), (3)\})$ is transformed into the termination problem $\{(2), (3)\}$ by a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}_1) = x_1$ and $\mathcal{P}ol(\mathsf{eval}_2) = x_1 - 1$ since

$$\forall i, j.\ i \geq 0 \Rightarrow i \geq 0$$
$$\forall i, j.\ i \geq 0 \Rightarrow i > i - 1$$
$$\forall i, j.\ j \leq i - 1 \Rightarrow i - 1 \geq i - 1$$
$$\forall i, j.\ j > i - 1 \Rightarrow i - 1 \geq i - 1$$

are true in the integers. The termination problem $\{(2), (3)\}$ is transformed into $\{(2)\}$ since (3) is not in the SCC of the problem's termination graph. This final termination problem can be handled using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}_2) = x_1 - x_2$ since

$$\forall i, j.\ j \leq i - 1 \Rightarrow i - j \geq 0$$
$$\forall i, j.\ j \leq i - 1 \Rightarrow i - j > i - (j + 1)$$

are true in the integers.

### C.9 First Example with Nondeterminism

This example stems from [4]. The imperative program fragment

```
while (x > 0 && y > 0) {
   either {
     (x, y) := (x - 1, ?);
   } or {
     y--;
   }
}
```

is translated into

$$\mathsf{eval}(x, y) \rightarrow \mathsf{eval}(x - 1, z)[\![x > 0 \wedge y > 0]\!] \quad (1)$$
$$\mathsf{eval}(x, y) \rightarrow \mathsf{eval}(x, y - 1)[\![x > 0 \wedge y > 0]\!] \quad (2)$$

The termination problem $\{(1), (2)\})$ is transformed into the termination problem $\{(2)\}$ by a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}) = x_1$ since

$$\forall x, y.\ x > 0 \wedge y > 0 \Rightarrow x \geq 0$$
$$\forall x, y.\ x > 0 \wedge y > 0 \Rightarrow x > x - 1$$
$$\forall x, y.\ x > 0 \wedge y > 0 \Rightarrow x \geq x$$

are true in the integers. The termination problem $\{(2)\})$ can be handled using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}) = x_2$ since

$$\forall x, y.\ x > 0 \wedge y > 0 \Rightarrow y \geq 0$$
$$\forall x, y.\ x > 0 \wedge y > 0 \Rightarrow y > y - 1$$

are true in the integers.

### C.10 Second Example with Nondeterminism

This example stems from [23]. The imperative program fragment

```
while (i - j >= 1) {
   (i, j) := (i - ?[? >= 0], j + ?[? > 0]);
}
```

is translated into

$$\mathsf{eval}(i, j) \to \mathsf{eval}(i - nat, j + pos)[\![i - j \geq 0 \wedge nat \geq 0 \wedge pos > 0]\!] \qquad (1)$$

The termination problem $\{(1)\}$) can be handled using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}) = x_1 - x_2$ since

$$\forall i, j, nat, pos.\ i - j \geq 0 \wedge nat \geq 0 \wedge pos > 0 \Rightarrow i - j \geq 0$$
$$\forall i, j, nat, pos.\ i - j \geq 0 \wedge nat \geq 0 \wedge pos > 0 \Rightarrow i - j > (i - nat) - (j + pos)$$

are true in the integers.

### C.11 Third Example with Nondeterminism

This example stems from [23]. The imperative program fragment

```
while (true) {
   if (x >= 0) {
      (x, y) := (x - 1, ?);
   } else if (y >= 0) {
      y--;
   } else {
      break;
   }
}
```

is translated into

$$\mathsf{eval}(x, y) \to \mathsf{eval}(x - 1, z)[\![x \geq 0]\!] \qquad (1)$$
$$\mathsf{eval}(x, y) \to \mathsf{eval}(x, y - 1)[\![y \geq 0]\!] \qquad (2)$$

The termination problem $\{(1), (2)\}$) is transformed into the termination problem $\{(2)\}$ by a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}) = x_1$ since

$$\forall x, y.\ x \geq 0 \Rightarrow x \geq 0$$
$$\forall x, y.\ x \geq 0 \Rightarrow x > x - 1$$
$$\forall x, y.\ y \geq 0 \Rightarrow x \geq x$$

are true in the integers. The termination problem $\{(2)\}$) can be handled using a $\mathcal{PA}$-polynomial interpretation with $\mathcal{P}ol(\mathsf{eval}) = x_2$ since

$$\forall x, y.\ y \geq 0 \Rightarrow y \geq 0$$
$$\forall x, y.\ y \geq 0 \Rightarrow y > y - 1$$

are true in the integers.

# D Detailed Empirical Results

| Example | Proof time (in s) | Time in `yices` (in s) | % of time in `yices` |
|---|---|---|---|
| A.1 | 0.042 | 0.029 | 69.05 |
| A.2 | 0.284 | 0.254 | 89.44 |
| A.3 | 0.318 | 0.281 | 88.36 |
| A.4 | 0.037 | 0.022 | 59.46 |
| A.5 | 0.014 | 0.010 | 71.43 |
| A.6 | 0.013 | 0.008 | 61.54 |
| A.7 | 0.012 | 0.008 | 66.67 |
| A.8 | 0.014 | 0.009 | 64.29 |
| A.9 | 0.011 | 0.007 | 63.64 |
| A.10 | 0.035 | 0.026 | 74.29 |
| A.11 | 0.057 | 0.045 | 78.95 |
| B.1 | 0.010 | 0.007 | 70.00 |
| B.2 | 0.010 | 0.007 | 70.00 |
| B.3 | 0.010 | 0.006 | 60.00 |
| B.4 | 0.007 | 0.007 | 100.00 |
| B.5 | 0.010 | 0.006 | 60.00 |
| B.6 | 0.010 | 0.006 | 60.00 |
| B.7 | 0.011 | 0.007 | 63.64 |
| B.8 | 0.166 | 0.154 | 92.77 |
| B.9 | 0.024 | 0.019 | 79.17 |
| B.10 | 0.030 | 0.022 | 73.33 |
| B.11 | 0.052 | 0.042 | 80.77 |
| B.12 | 0.060 | 0.050 | 83.33 |
| B.13 | 0.059 | 0.049 | 83.05 |
| B.14 | 0.031 | 0.026 | 83.87 |
| B.15 | 0.032 | 0.026 | 81.25 |
| B.16 | 0.039 | 0.029 | 74.36 |
| B.17 | 0.040 | 0.030 | 75.00 |
| B.18 | 0.070 | 0.059 | 84.29 |
| C.1 | 0.043 | 0.030 | 69.77 |
| C.2 | 0.042 | 0.030 | 71.43 |
| C.3 | 0.034 | 0.025 | 73.53 |
| C.4 | 0.036 | 0.030 | 83.33 |
| C.5 | 0.032 | 0.027 | 84.38 |
| C.6 | 0.086 | 0.070 | 81.40 |
| C.7 | 0.012 | 0.007 | 58.33 |
| C.8 | 0.043 | 0.034 | 79.07 |
| C.9 | 0.031 | 0.023 | 74.19 |
| C.10 | 0.011 | 0.007 | 63.64 |
| C.11 | 0.032 | 0.023 | 71.88 |

## References


1. Thomas Arts and Jürgen Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236(1–2):133–178, 2000.
2. Aaron Bradley, Zohar Manna, and Henny Sipma. Linear ranking with reachability. In Kousha Etessami and Sriram Rajamani, editors, *Proceedings of the 17th Conference on Computer Aided Verification (CAV '05)*, volume 3576 of *Lecture Notes in Computer Science*, pages 491–504. Springer-Verlag, 2005.
3. Aaron Bradley, Zohar Manna, and Henny Sipma. Termination of polynomial programs. In Radhia Cousot, editor, *Proceedings of the 6th Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI '05)*, volume 3385 of *Lecture Notes in Computer Science*, pages 113–129. Springer-Verlag, 2005.
4. Aziem Chawdhary, Byron Cook, Sumit Gulwani, Mooly Sagiv, and Hongseok Yang. Ranking abstractions. In Sophia Drossopoulou, editor, *Proceedings of the 17th European Symposium on Programming (ESOP '08)*, volume 4960 of *Lecture Notes in Computer Science*, pages 148–162. Springer-Verlag, 2008.
5. Michael Colón and Henny Sipma. Synthesis of linear ranking functions. In Tiziana Margaria and Wang Yi, editors, *Proceedings of the 7th Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '01)*, volume 2031 of *Lecture Notes in Computer Science*, pages 67–81. Springer-Verlag, 2001.
6. Michael Colón and Henny Sipma. Practical methods for proving program termination. In Ed Brinksma and Kim Guldstrand Larsen, editors, *Proceedings of the 14th Conference on Computer Aided Verification (CAV '02)*, volume 2404 of *Lecture Notes in Computer Science*, pages 442–454. Springer-Verlag, 2002.
7. Evelyne Contejean, Claude Marché, Ana Paula Tomás, and Xavier Urbain. Mechanically proving termination using polynomial interpretations. *Journal of Automated Reasoning*, 34(4):325–363, 2005.
8. Byron Cook, Andreas Podelski, and Andrey Rybalchenko. Abstraction refinement for termination. In Chris Hankin and Igor Siveroni, editors, *Proceedings of the 12th Symposium on Static Analysis (SAS '05)*, volume 3672 of *Lecture Notes in Computer Science*, pages 87–101. Springer-Verlag, 2005.
9. Byron Cook, Andreas Podelski, and Andrey Rybalchenko. Termination proofs for systems code. In *Proceedings of the 2006 Conference on Programming Language Design and Implementation (PLDI '06)*, pages 415–426, 2006.
10. Byron Cook, Andreas Podelski, and Andrey Rybalchenko. Terminator: Beyond safety. In Thomas Ball and Robert Jones, editors, *Proceedings of the 18th Conference on Computer Aided Verification (CAV '06)*, volume 4144 of *Lecture Notes in Computer Science*, pages 415–418. Springer-Verlag, 2006.
11. Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th Symposium on Principles of Programming Languages (POPL '77)*, pages 238–252, 1977.
12. Patrick Cousot and Nicolas Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Proceedings of the 5th Symposium on Principles of Programming Languages (POPL '78)*, pages 84–96, 1978.
13. Stephan Falke and Deepak Kapur. Dependency pairs for rewriting with built-in numbers and semantic data structures. In Andrei Voronkov, editor, *Proceedings of the 19th International Conference on Rewriting Techniques and Applications (RTA '08)*, volume 5117 of *Lecture Notes in Computer Science*, pages 94–109. Springer-Verlag, 2008. An expanded version is Technical Report TR-CS-2007-21, available from `http://www.cs.unm.edu/research/tech-reports/`.

14. Carsten Fuhs, Jürgen Giesl, Aart Middeldorp, Peter Schneider-Kamp, René Thiemann, and Harald Zankl. SAT solving for termination analysis with polynomial interpretations. In João Marques-Silva and Karem Sakallah, editors, *Proceedings of the 10th Conference on Theory and Applications of Satisfiability Testing (SAT '05)*, volume 4501 of *Lecture Notes in Computer Science*, pages 340–354. Springer-Verlag, 2007. An expanded version is Technical Report AIB-2007-02, available from `http://aib.informatik.rwth-aachen.de/`.
15. Jürgen Giesl, René Thiemann, and Peter Schneider-Kamp. The dependency pair framework: Combining techniques for automated termination proofs. In Franz Baader and Andrei Voronkov, editors, *Proceedings of the 11th Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR '04)*, volume 3452 of *Lecture Notes in Artificial Intelligence*, pages 301–331. Springer-Verlag, 2005.
16. Jürgen Giesl, René Thiemann, Stephan Swiderski, and Peter Schneider-Kamp. Proving termination by bounded increase. In Frank Pfenning, editor, *Proceedings of the 21st Conference on Automated Deduction (CADE '07)*, volume 4603 of *Lecture Notes in Artificial Intelligence*, pages 443–459. Springer-Verlag, 2007. An expanded version is Technical Report AIB-2007-03, available from `http://aib.informatik.rwth-aachen.de/`.
17. Nao Hirokawa and Aart Middeldorp. Tyrolean termination tool: Techniques and features. *Information and Computation*, 205(4):474–511, 2007.
18. Hoon Hong and Dalibor Jakuš. Testing positiveness of polynomials. *Journal of Automated Reasoning*, 21(1):23–38, 1998.
19. Keiichirou Kusakari, Masaki Nakamura, and Yoshihito Toyama. Argument filtering transformation. In Gopalan Nadathur, editor, *Proceedings of the 1st Conference on Principles and Practice of Declarative Programming (PPDP '99)*, volume 1702 of *Lecture Notes in Computer Science*, pages 47–61. Springer-Verlag, 1999.
20. Dallas Lankford. On proving term rewriting systems are Noetherian. Memo MTP-3, Mathematics Department, Louisiana Tech University, Ruston, 1979.
21. Antoine Miné. The octagon abstract domain. *Higher-Order and Symbolic Computation*, 19(1):31–100, 2006.
22. Gerald Peterson and Mark Stickel. Complete sets of reductions for some equational theories. *Journal of the ACM*, 28(2):233–264, 1981.
23. Andreas Podelski and Andrey Rybalchenko. A complete method for the synthesis of linear ranking functions. In Bernhard Steffen and Giorgio Levi, editors, *Proceedings of the 5th Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI '04)*, volume 2937 of *Lecture Notes in Computer Science*, pages 239–251. Springer-Verlag, 2004.
24. Andreas Podelski and Andrey Rybalchenko. Transition invariants. In *Proceedings of the 19th Symposium on Logic in Computer Science (LICS '04)*, pages 32–41. IEEE Computer Society, 2004.
25. Mojzesz Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In *Comptes Rendus du Premier Congrès de Mathématiciens des Pays Slaves*, pages 92–101, 1929.
26. Ashish Tiwari. Termination of linear programs. In Rajeev Alur and Doron Peled, editors, *Proceedings of the 16th Conference on Computer Aided Verification (CAV '04)*, volume 3114 of *Lecture Notes in Computer Science*, pages 70–82. Springer-Verlag, 2004.
27. Hans Zantema. Termination. In TeReSe, editor, *Term Rewriting Systems*, chapter 6. Cambridge University Press, 2003.