# Transforming Dependency Chains of Constrained TRSs into Bounded Monotone Sequences of Integers

Tomohiro Sasano
Graduate School of Information Science
Nagoya University
Nagoya, Japan
sasano@trs.cm.is.nagoya-u.ac.jp

Naoki Nishida
Graduate School of Informatics
Nagoya University
Nagoya, Japan
nishida@i.nagoya-u.ac.jp

Masahiko Sakai
Graduate School of Informatics
Nagoya University
Nagoya, Japan
sakai@i.nagoya-u.ac.jp

Tomoya Ueyama
Graduate School of Information Science
Nagoya University
Nagoya, Japan

In the dependency pair framework for proving termination of rewriting systems, polynomial interpretations are used to transform dependency chains into bounded decreasing sequences of integers, and they play an important role for the success of proving termination, especially for constrained rewriting systems. In this paper, we show sufficient conditions of linear polynomial interpretations for transforming dependency chains into bounded monotone (i.e., decreasing or increasing) sequences of integers. Such polynomial interpretations transform rewrite sequences of the original system into decreasing or increasing sequences independently of the transformation of dependency chains. When we transform rewrite sequences into increasing sequences, polynomial interpretations have negative coefficients for reducible positions of marked function symbols. We propose four DP processors parametrized by transforming dependency chains and rewrite sequences into either decreasing or increasing sequences of integers, respectively. We show that such polynomial interpretations make us succeed in proving termination of the McCarthy 91 function over the integers.

## 1 Introduction

Recently, techniques developed for term rewriting systems (TRSs, for short) have been applied to the verification of programs written in several programming languages (cf. [10]). In verifying programs with comparison operators over the integers via term rewriting, *constrained rewriting* is very useful to avoid very complicated rewrite rules for the comparison operators, and various formalizations of constrained rewriting have been proposed: *constrained TRSs* [11, 4, 23, 22] (e.g., *membership conditional TRSs* [24]), *constrained equational systems* (CESs, for short) [5], *integer TRSs* (ITRSs, for short) [9], *PA-based TRSs* ($\mathbb{Z}$-TRSs) [6] (simplified variants of CESs), and *logically constrained TRSs* (LCTRSs, for short) [16, 17].

One of the most important properties that are often verified in practice is *termination*, and many methods for proving termination have been developed in the literature, especially in the field of term rewriting (cf. the survey of Zantema [25]). At present, the *dependency pair* (DP) *method* [2] and the *DP framework* [13] are key fundamentals for proving termination of TRSs, and they have been extended to several kinds of rewrite systems [5, 1, 6, 9, 22, 19, 10]. In the DP framework, termination problems are reduced to finiteness of *DP problems* which consist of sets of dependency pairs and rewrite rules. We prove finiteness by applying *sound DP processors* to an input DP problem and then by decomposing the DP problem to smaller ones in the sense that all the DP sets of output DP problems are strict subsets of the DP set of the input problem. In the DP frameworks [5, 22] for constrained rewriting, the DP processors based on polynomial interpretations (the PI-based processors, for short) decompose a given DP problem by using a polynomial interpretation $\mathcal{P}ol$ that transforms dependency chains into bounded

Submitted to:
WPTE 2017

decreasing sequences of integers—roughly speaking, a dependency pair $s^\sharp \to t^\sharp$ $[\![\varphi]\!]$ is removed from the given problem if the integer arithmetic formula $\varphi \Rightarrow \mathcal{P}ol(s) > \mathcal{P}ol(t)$ is valid. The processor in [22] can be considered a simplified version of that in [5] in the sense that for efficiency, $\mathcal{P}ol$ drops *reducible positions*—arguments of marked symbols, which may contain an uninterpreted function symbol when a dependency pair is instantiated—and then the rules in the given system are ignored. Such a simplification is not so restrictive when we prove termination of *counter-controlled loops*, e.g., `for(` $i=0$`;` $i<n$`;` $i$`++){...}`. However, the simplification sometimes prevents us from proving termination of a function, the definition of which has nested function calls.

Let us consider the following constrained TRS defining the *McCarthy 91 function*:

$$\mathcal{R}_1 = \left\{ \begin{array}{lll} (1) & \mathsf{f}(x) \to \mathsf{f}(\mathsf{f}(\mathsf{s}^{11}(x))) & [\![\ \mathsf{s}^{101}(\mathsf{0}) > x\ ]\!] \\ (2) & \mathsf{f}(x) \to \mathsf{p}^{10}(x) & [\![\neg(\mathsf{s}^{101}(\mathsf{0}) > x)]\!] \end{array} \right\} \cup \left\{ \begin{array}{l} \mathsf{s}(\mathsf{p}(x)) \to x \\ \mathsf{p}(\mathsf{s}(x)) \to x \end{array} \right\}$$

It is known that the function always terminates and returns 91 if an integer $n \leq 101$ is given as input, and $n-10$ otherwise: $\forall n \in \mathbb{Z}.\ (n \leq 101 \Rightarrow \mathsf{f}(n) = 91) \wedge (n > 101 \Rightarrow \mathsf{f}(n) = n-10)$. Termination of the McCarthy 91 function can be proved automatically if the function is defined over the natural numbers [12]. However, the method in [12] cannot prove termination of the function that is defined over the integers. As another approach, let us consider the DP framework. The dependency pairs of $\mathcal{R}_1$ are:

$$DP(\mathcal{R}_1) = \left\{ \begin{array}{lll} (3) & \mathsf{f}^\sharp(x) \to \mathsf{f}^\sharp(\mathsf{f}(\mathsf{s}^{11}(x))) & [\![\mathsf{s}^{101}(\mathsf{0}) > x]\!] \\ (4) & \mathsf{f}^\sharp(x) \to \mathsf{f}^\sharp(\mathsf{s}^{11}(x)) & [\![\mathsf{s}^{101}(\mathsf{0}) > x]\!] \end{array} \right\}$$

Unfortunately, the method in [22] for proving termination of constrained TRSs cannot prove termination of $\mathcal{R}_1$ because the right-hand side of (3) is of the form $\mathsf{f}^\sharp(\mathsf{f}(\mathsf{s}^{11}(x)))$ and thus we have to drop the first argument of $f^\sharp$, i.e., $\mathcal{P}ol(f^\sharp) = a_0$ where $a_0$ is an integer—both sides of (3) and (4) are converted by $\mathcal{P}ol$ to $a_0$ and we do not remove any of (3) and (4). To make the method in [22] more powerful, let us allow $\mathcal{P}ol(f^\sharp)$ to keep its reducible positions as in [5]. Then, for $\mathcal{R}_1$, $\mathcal{P}ol$ has to be an interpretation over the natural numbers, and for each rule $\ell \to r$ $[\![\varphi]\!]$ in $\mathcal{R}_1$ the validity of the integer arithmetic formula $\varphi \Rightarrow \mathcal{P}ol(\ell) \geq \mathcal{P}ol(r)$ is required. However, such an interpretation does not exist for $\mathcal{R}_1$. Note that $\mathsf{Ctrl}$ [18] fails to prove termination of the LCTRS corresponding to $\mathcal{R}_1$.

In this paper, we extend the PI-based processor in [22] by making its linear polynomial interpretation $\mathcal{P}ol$ transform dependency chains into bounded *monotone* (i.e., decreasing or increasing) sequences of integers. To be more precise, given a constrained TRS $\mathcal{R}$,

- $\mathcal{P}ol$ is an interpretation over the natural numbers as in [5], while constants that are not coefficients may be negative integers (i.e., for $\mathcal{P}ol(f) = b_0 + b_1x_1 + \cdots + b_nx_n$, the coefficients $b_1, \ldots, b_n$ have to be positive integers but the constant $b_0$ may be a negative integer),
- for rules in $\mathcal{R}$, we require one of the following:
  (**R1**) $\varphi \Rightarrow \mathcal{P}ol(\ell) \geq \mathcal{P}ol(r)$ is valid for all $\ell \to r$ $[\![\varphi]\!] \in \mathcal{R}$ (i.e., rewrite sequences of $\mathcal{R}$ are transformed by $\mathcal{P}ol$ into decreasing sequences of integers), or
  (**R2**) $\varphi \Rightarrow \mathcal{P}ol(\ell) \leq \mathcal{P}ol(r)$ is valid for all $\ell \to r$ $[\![\varphi]\!] \in \mathcal{R}$ (i.e., rewrite sequences of $\mathcal{R}$ are transformed by $\mathcal{P}ol$ into increasing sequences of integers), and
- for monotonicity of transformed sequences, coefficients for reducible positions have to satisfy a sufficient condition—to be non-negative for (**R1**) and to be negative for (**R2**)—and the second argument of the subtraction symbol (i.e., $-$) is an interpretable term in anywhere.

Such a polynomial interpretation transforms all dependency chains into bounded decreasing sequences of integers, or all to bounded increasing sequences of integers. Since we have two possibilities for transforming rewrite sequences of $\mathcal{R}$, we have four kinds of PI-based processors. Then, we show an experimental result to compare the four PI-based processors by using them to prove termination of $\mathcal{R}_1$ shown in this section. Although this paper adopts the class of constrained TRSs in [11, 23, 22], it would be straightforward to adapt our results to other higher-level styles of constrained systems in, e.g., [5, 7, 16]. It would also be straightforward to extend the results for the single-sorted case to the many-sorted one (cf. [15]).

The contribution of this paper is to formalize linear polynomial interpretations that transform dependency chains into bounded monotone (i.e., not only decreasing but also increasing) sequences of integers, and that transform rewrite sequences of the given constrained TRS into monotone sequences of integers.

## 2 Preliminaries

In this section, we briefly recall the basic notions and notations of term rewriting [3, 21], and constrained rewriting [11, 4, 23, 22].

Throughout the paper, we use $\mathcal{V}$ as a countably infinite set of *variables*. We denote the set of *terms* over a signature $\Sigma$ and a variable set $V \subseteq \mathcal{V}$ by $T(\Sigma, V)$. We often write $f/n$ to represent an $n$-ary symbol $f$. We abbreviate the set $T(\Sigma, \emptyset)$ of *ground terms* over $\Sigma$ to $T(\Sigma)$. We denote the set of variables appearing in a term $t$ by $\mathcal{V}ar(t)$. A *hole* $\Box$ is a special constant not appearing in considered signatures (i.e., $\Box \notin \Sigma$), and a term in $T(\Sigma \cup \{\Box\}, V)$ is called a *context* over $\Sigma$ and $V$ if the hole $\Box$ appears in the term exactly once. We denote the set of contexts over $\Sigma$ and $V$ by $T_\Box(\Sigma, V)$. For a term $t$ and a context $C[\,]_p$ with the hole at a *position* $p$, we denote by $C[t]_p$ the term obtained from $t$ and $C[\,]_p$ by replacing the hole at $p$ by $t$. We may omit $p$ from $C[\,]_p$ and $C[t]_p$. For a term $C[t]_p$, the term $t$ is a *subterm* of $C[t]$ (at $p$). Especially, when $p$ is not the *root* position $\varepsilon$, we call $t$ a *proper subterm* of $C[t]$. For a term $s$ and a position $p$ of $s$, we denote the subterm of $s$ at $p$ by $s|_p$, and the function symbol at the root position of $s$ by $root(s)$.

The *domain* and *range* of a *substitution* $\sigma$ are denoted by $\mathcal{D}om(\sigma)$ and $\mathcal{R}an(\sigma)$, respectively. For a signature $\Sigma$, a substitution $\sigma$ is called *ground* if $\mathcal{R}an(\sigma) \subseteq T(\Sigma)$. For a subset $V$ of $\mathcal{V}$, we denote the set of substitutions over $\Sigma$ and $V$ by $\mathcal{S}ub(\Sigma, V)$: $\mathcal{S}ub(\Sigma, V) = \{\sigma \mid \mathcal{R}an(\sigma) \subseteq T(\Sigma, V)\}$. We abbreviate $\mathcal{S}ub(\Sigma, \emptyset)$ to $\mathcal{S}ub(\Sigma)$. We may write $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ instead of $\sigma$ if $\mathcal{D}om(\sigma) = \{x_1, \ldots, x_n\}$ and $\sigma(x_i) = t_i$ for all $1 \le i \le n$. We may write $t\sigma$ for the application $\sigma(t)$ of $\sigma$ to $t$. For a subset $V$ of $\mathcal{V}$, we denote the restricted substitution of $\sigma$ w.r.t. $V$ by $\sigma|_V$: $\sigma|_V = \{x \mapsto \sigma(x) \mid x \in \mathcal{D}om(\sigma) \cap V\}$.

Let $\mathcal{G}$ be a signature (e.g., a subsignature of $\Sigma$) and $\mathcal{P}$ a set of *predicate symbols*, each of which has a fixed arity, and $\mathcal{M}$ a *structure* specifying interpretations for symbols in $\mathcal{G}$ and $\mathcal{P}$: $\mathcal{M}$ has a *universe* (a non-empty set), and $g^{\mathcal{M}}$ and $p^{\mathcal{M}}$ are interpretations for a function symbol $g \in \mathcal{G}$ and a predicate symbol $p \in \mathcal{P}$, respectively. Ground terms in $T(\mathcal{G})$ are interpreted by $\mathcal{M}$ in the usual way. We use $\top$ and $\bot$ for Boolean values *true* and *false*,[1] and usual logical connectives $\neg$, $\vee$, $\wedge$, and $\Rightarrow$, which are interpreted in the usual way. For the sake of simplicity, we do not use quantifiers in formulas. We assume that $\mathcal{P}$ contains a binary symbol $\simeq$ for *equality*. For a subset $V \subseteq \mathcal{V}$, we denote the set of *formulas* over $\mathcal{G}$, $\mathcal{P}$, and $V$ by $\mathcal{F}ol(\mathcal{G}, \mathcal{P}, V)$. The set of variables in a formula $\varphi$ is denoted by $\mathcal{V}ar(\varphi)$. Formulas in $\mathcal{F}ol(\mathcal{G}, \mathcal{P}, \mathcal{V})$ are called *constraints* (w.r.t. $\mathcal{M}$). We assume that for each element $a$ in the universe, there exists a ground term $t$ in $T(\mathcal{G})$ such that $t^{\mathcal{M}} = a$. A ground formula $\varphi$ is said to *hold w.r.t* $\mathcal{M}$, written as $\mathcal{M} \models \varphi$, if $\varphi$ is interpreted by $\mathcal{M}$ as *true*. The application of a substitution $\sigma \in \mathcal{S}ub(\mathcal{G}, \mathcal{V})$ is naturally extended to formulas, and $\sigma(\varphi)$ is abbreviated to $\varphi\sigma$. Note that for a signature $\Sigma$ with $\mathcal{G} \subseteq \Sigma$, we cannot apply $\sigma$ to $\varphi \in \mathcal{F}ol(\mathcal{G}, \mathcal{P}, \mathcal{V})$ if $\sigma|_{\mathcal{V}ar(\varphi)} \notin \mathcal{S}ub(\mathcal{G}, \mathcal{V})$.[2] A formula $\varphi$ is called *valid w.r.t.* $\mathcal{M}$ ($\mathcal{M}$-*valid*, for short)

[1]Note that $\top$ and $\bot$ are just symbols used in e.g., constraints of rewrite rules, and we distinguish them with *true* and *false* used as values.

[2]When considering formulas in $\mathcal{F}ol(\mathcal{G}, \mathcal{P}, \mathcal{V})$, we force $\sigma\varphi$ to be in $\mathcal{F}ol(\mathcal{G}, \mathcal{P}, \mathcal{V})$.

if $\mathcal{M} \models \varphi\sigma$ for all ground substitutions $\sigma \in \mathcal{S}ub(\mathcal{G})$ with $\mathcal{V}ar(\varphi) \subseteq \mathcal{D}om(\sigma)$, and called *satisfiable w.r.t.* $\mathcal{M}$ (*$\mathcal{M}$-satisfiable*, for short) if $\mathcal{M} \models \varphi\sigma$ for some ground substitution $\sigma \in \mathcal{S}ub(\mathcal{G})$ such that $\mathcal{V}ar(\varphi) \subseteq \mathcal{D}om(\sigma)$. A structure $\mathcal{M}$ for $\mathcal{G}$ and $\mathcal{P}$ is called an *LIA-structure* if the universe is the integers, every symbol $g \in \mathcal{G}$ is interpreted as a linear integer arithmetic expression, and every symbol $p \in \mathcal{P}$ is interpreted as a Presburger arithmetic sentence over the integers, e.g., binary comparison predicates.

Let $\mathcal{F}$ and $\mathcal{G}$ be pairwise disjoint signatures (i.e., $\mathcal{F} \cap \mathcal{G} = \emptyset$),[3] $\mathcal{P}$ a set of predicate symbols, and $\mathcal{M}$ a structure for $\mathcal{G}$ and $\mathcal{P}$. A *constrained rewrite rule* over $(\mathcal{F},\mathcal{G},\mathcal{P},\mathcal{M})$ is a triple $(\ell, r, \varphi)$, denoted by $\ell \to r \; [\![\varphi]\!]$, such that $\ell, r \in T(\mathcal{F} \cup \mathcal{G}, \mathcal{V})$, $\ell$ is not a variable, $\varphi \in \mathcal{F}ol(\mathcal{G},\mathcal{P},\mathcal{V})$, and $\mathcal{V}ar(\ell) \supseteq \mathcal{V}ar(r) \cup \mathcal{V}ar(\varphi)$. We usually consider $\mathcal{M}$-satisfiable constraints for $\varphi$. When $\varphi$ is $\top$, we may write $\ell \to r$ instead of $\ell \to r \; [\![\top]\!]$. A *constrained term rewriting system* (constrained TRS, for short) over $(\mathcal{F},\mathcal{G},\mathcal{P},\mathcal{M})$ is a finite set $\mathcal{R}$ of constrained rewrite rules over $(\mathcal{F},\mathcal{G},\mathcal{P})$. When $\varphi = \top$ for all rules $\ell \to r \; [\![\varphi]\!] \in \mathcal{R}$, $\mathcal{R}$ is a *term rewriting system* (TRS, for short). The *rewrite relation* $\to_{\mathcal{R}}$ of $\mathcal{R}$ is defined as follows: $\to_{\mathcal{R}} = \{(C[\ell\sigma]_p, C[r\sigma]_p) \mid \ell \to r \; [\![\varphi]\!] \in \mathcal{R},\ C[\;] \in T_{\square}(\mathcal{F} \cup \mathcal{G}, \mathcal{V}),\ \sigma \in \mathcal{S}ub(\mathcal{F} \cup \mathcal{G}, \mathcal{V}),\ \sigma|_{\mathcal{V}ar(\varphi)} \in \mathcal{S}ub(\mathcal{G},\mathcal{V}),\ \varphi\sigma \text{ is } \mathcal{M}\text{-valid}\}$. To specify the position $p$ where the term is reduced, we may write $\to_{p,\mathcal{R}}$ or $\to_{>q,\mathcal{R}}$ where $p > q$. A term $t$ is called *terminating* if there is no infinite reduction sequence $t \to_{\mathcal{R}} t_1 \to_{\mathcal{R}} t_2 \to_{\mathcal{R}} \cdots$. $\mathcal{R}$ is called *terminating* if every term is terminating. For a constrained TRS $\mathcal{R}$ over $(\mathcal{F},\mathcal{G},\mathcal{P},\mathcal{M})$, the sets $\mathcal{D}_{\mathcal{R}}$ and $\mathcal{C}_{\mathcal{R}}$ of *defined symbols* and *constructors*, respectively, are defined as follows: $\mathcal{D}_{\mathcal{R}} = \{f \in \mathcal{F} \cup \mathcal{G} \mid f(t_1,\ldots,t_n) \to r \; [\![\varphi]\!] \in \mathcal{R}\}$ and $\mathcal{C}_{\mathcal{R}} = (\mathcal{F} \cup \mathcal{G}) \setminus \mathcal{D}_{\mathcal{R}}$.

*Example 2.1* Let $\mathcal{G}_{\mathsf{LIA}} = \{\mathsf{0}/0, \mathsf{s}/1, \mathsf{p}/1\}$, $\mathcal{P}_{\mathsf{LIA}} = \{\simeq, >, \geq\}$, and $\mathcal{M}_{\mathsf{LIA}}$ an LIA-structure for $\mathcal{G}_{\mathsf{LIA}}$ and $\mathcal{P}_{\mathsf{LIA}}$ such that the universe is $\mathbb{Z}$, $\mathsf{0}^{\mathcal{M}_{\mathsf{LIA}}} = 0$, $\mathsf{s}^{\mathcal{M}_{\mathsf{LIA}}}(x) = x+1$, $\mathsf{p}^{\mathcal{M}_{\mathsf{LIA}}}(x) = x-1$, and $>$ and $\geq$ are interpreted as the corresponding comparison predicates in the usual way. Then, we have that $(\mathsf{s}(\mathsf{s}(\mathsf{0})))^{\mathcal{M}_{\mathsf{LIA}}} = 2$, $(\mathsf{s}(\mathsf{p}(\mathsf{p}(\mathsf{s}(\mathsf{0})))))^{\mathcal{M}_{\mathsf{LIA}}} = 0$, and so on. $\mathcal{R}_1$ in Section 1 is over $(\{\mathsf{f}/1\}, \mathcal{G}_{\mathsf{LIA}}, \mathcal{P}_{\mathsf{LIA}}, \mathcal{M}_{\mathsf{LIA}})$, and we have e.g., $\mathsf{f}(\mathsf{s}^{100}(\mathsf{0})) \to_{\mathcal{R}_1} \mathsf{f}(\mathsf{f}(\mathsf{s}^{111}(\mathsf{0}))) \to_{\mathcal{R}_1} \mathsf{f}(\mathsf{p}^{10}(\mathsf{s}^{111}(\mathsf{0}))) \to^*_{\mathcal{R}_1} \mathsf{f}(\mathsf{s}^{101}(\mathsf{0})) \to_{\mathcal{R}_1} \mathsf{p}^{10}(\mathsf{s}^{101}(\mathsf{0})) \to^*_{\mathcal{R}_1} \mathsf{s}^{91}(\mathsf{0})$.

We assume that $\mathcal{R}$ is *locally sound for* $\mathcal{M}$ [23, 22], i.e., for every rule $\ell \to r \; [\![\varphi]\!] \in \mathcal{R}$, if the root symbol of $\ell$ is in $\mathcal{G}$, then $r$ and all the proper subterms of $\ell$ are in $T(\mathcal{G},\mathcal{V})$, and the formula $\varphi \Rightarrow (\ell \simeq r)$ is $\mathcal{M}$-valid. Local soundness for $\mathcal{M}$ ensures consistency for the semantics and further that no interpreted ground term is reduced to any term containing an uninterpreted function symbol. This property is implicitly assumed in other formalizations of constrained rewriting by e.g., rules for constructors are separated from user-defined rules [4, 5], or rules are defined for uninterpreted function symbols only [16].

## 3 The DP Framework for Constrained TRSs

In this section, we recall the *DP framework* for constrained TRSs [22], which is a straightforward extension of the DP framework [13, 5] for TRSs to constrained TRSs.

In the following, we let $\mathcal{R}$ be a constrained TRS over $(\mathcal{F},\mathcal{G},\mathcal{P},\mathcal{M})$ unless noted otherwise. We introduce a *marked symbol* $f^\sharp$ for each defined symbol $f$ of $\mathcal{R}$, where $f^\sharp \notin \mathcal{F} \cup \mathcal{G}$. We denote the set of marked symbol by $\mathcal{D}^\sharp_{\mathcal{R}}$. For a term $t$ of the form $f(t_1,\ldots,t_n)$ in $T(\mathcal{F} \cup \mathcal{G}, \mathcal{V})$ with $f/n \in \mathcal{D}_{\mathcal{R}}$, we denote $f^\sharp(t_1,\ldots,t_n)$ (a marked term) by $t^\sharp$. To make it clear whether a term is marked, we often attach explicitly the mark $\sharp$ to meta variables for marked terms. A *constrained marked pair* over $(\mathcal{F} \cup \mathcal{D}^\sharp_{\mathcal{R}}, \mathcal{G}, \mathcal{P})$ is a triple $(s^\sharp, t^\sharp, \varphi)$, denoted by $s^\sharp \to t^\sharp \; [\![\varphi]\!]$, such that $s$ and $t$ are terms in $T(\mathcal{F} \cup \mathcal{G}, \mathcal{V})$, both $s$ and $t$ are rooted by defined symbols of $\mathcal{R}$, and $\mathcal{V}ar(s) \supseteq \mathcal{V}ar(t) \cup \mathcal{V}ar(\varphi)$. When $\varphi$ is $\top$, we may write $s^\sharp \to t^\sharp$ instead of $s^\sharp \to t^\sharp \; [\![\varphi]\!]$. A constrained marked pair $s^\sharp \to t^\sharp \; [\![\varphi]\!]$ is called a *dependency pair* of $\mathcal{R}$ if there exists

[3] A signature $\Sigma$ is explicitly divided into $\mathcal{F}$ and $\mathcal{G}$ (i.e., $\Sigma = \mathcal{F} \uplus \mathcal{G}$) where $\mathcal{F}$ is the set of *uninterpreted* symbols and $\mathcal{G}$ the set of *interpreted* symbols. To make this distinguish clear, we always separate $\mathcal{F}$ and $\mathcal{G}$, e.g., we write $(\mathcal{F},\mathcal{G})$ but not $\mathcal{F} \uplus \mathcal{G}$.

a renamed variant $s \to C[t]$ $[\![ \varphi ]\!]$ of a rewrite rule in $\mathcal{R}$. We denote the set of dependency pairs of $\mathcal{R}$ by $DP(\mathcal{R})$. In the following, we let $\mathcal{S}$ be a set of dependency pairs related to $\mathcal{R}$ unless noted otherwise.

A (possibly infinite) derivation $s_0^\sharp\sigma_0 \to_{\varepsilon,\mathcal{S}} t_0^\sharp\sigma_0 \to^*_{>\varepsilon,\mathcal{R}} s_1^\sharp\sigma_1 \to_{\varepsilon,\mathcal{S}} t_1^\sharp\sigma_1 \to^*_{>\varepsilon,\mathcal{R}} \cdots$ with $\sigma_0,\sigma_1,\sigma_2,\ldots \in \mathcal{S}ub(\mathcal{F}\cup\mathcal{G},\mathcal{V})$ is called a *dependency chain w.r.t.* $\mathcal{S}$ ($\mathcal{S}$-chain, for short). The chain is called *infinite* if it contains infinitely many $\to_{\varepsilon,\mathcal{S}}$ steps, and called *minimal* if $t_i^\sharp\sigma_i$ is terminating w.r.t. $\mathcal{R}$ for all $i \geq 0$. We deal with "minimal chains" only, and chains in this paper are minimal unless noted otherwise.

**Theorem 3.1 ([22])** *$\mathcal{R}$ is terminating iff there is no infinite $DP(\mathcal{R})$-chain.*

A pair $(\mathcal{S},\mathcal{R})$ of sets of dependency pairs and constrained rewrite rules is called a *DP problem*. We denote a DP problem $(\mathcal{S},\mathcal{R})$ by $\mathcal{S}$ because in this paper, we do not modify $\mathcal{R}$. A DP problem $\mathcal{S}$ is called *finite* if there is no infinite $\mathcal{S}$-chain, and called *infinite* if the DP problem is not finite or $\mathcal{R}$ is not terminating. Note that there are DP problems which are both finite and infinite (see [14]). A DP problem $\mathcal{S}$ is called *trivial* if $\mathcal{S} = \emptyset$. A *DP processor* is a function which takes a DP problem as input and returns a finite set of DP problems. A DP processor *Proc* is called *sound* if for any DP problem $\mathcal{S}$, the DP problem is finite whenever all the DP problems in $Proc(\mathcal{S})$ are finite. *Proc* is called *complete* if for any DP problem $\mathcal{S}$, the DP problem is infinite whenever there exists an infinite DP problem in $Proc(\mathcal{S})$. The *DP framework* is a method to prove/disprove the finiteness of DP problems:[4] given a constrained TRS $\mathcal{R}$, if the *initial* DP problem $DP(\mathcal{R})$ is decomposed into trivial DP problems by sound DP processors, then the framework succeeds in proving termination of $\mathcal{R}$.

In the rest of this section, we briefly introduce the DP processor based on *polynomial interpretations* (PI, for short), which is an extension of those in the DP framework for TRSs.

The *PI-based* processor in [22] is defined for constrained TRSs with an LIA-structure $\mathcal{M}_\mathbb{Z}$ with binary predicate symbols $>$ and $\geq$. Given a signature $\Sigma = \mathcal{F} \uplus \mathcal{G}_\mathbb{Z}$ with $\mathcal{G}_\mathbb{Z} \supseteq \{+,-\}$, we define a *linearpolynomial interpretation $\mathcal{P}ol$ for a subsignature $\mathcal{F}' \subseteq \mathcal{F}$*, $n$-ary function symbol $f$ in $\mathcal{F}'$, $\mathcal{P}ol(f)$ is a term in $T(\mathcal{G}_\mathbb{Z},\{x_1,\ldots,x_n\})$ that represents a linear polynomial. Note that $\mathcal{G}_\mathbb{Z}$ and $\mathcal{M}_\mathbb{Z}$ may be different from $\mathcal{G}_{\mathsf{LIA}}$ and $\mathcal{M}_{\mathsf{LIA}}$ in Example 2.1. For readability, we use usual mathematical notions for terms in $T(\mathcal{G}_\mathbb{Z},\mathcal{V})$—e.g., 100 for $\mathsf{s}^{100}(0)$, $2x$ for $x+x$, and so on—and, given an $n$-ary symbol $f$ in $\mathcal{F}'$, we write $a_0 + a_1x_1 + \cdots + a_nx_n$ for $\mathcal{P}ol(f)$. We apply $\mathcal{P}ol$ for $\mathcal{F}'$ to arbitrary terms in $T(\mathcal{F}\cup\mathcal{G}_\mathbb{Z},\mathcal{V})$ as follows: $\mathcal{P}ol(x) = x$ for $x \in \mathcal{V}$; $\mathcal{P}ol(f(t_1,\ldots,t_n)) = \mathcal{P}ol(f)\{x_i \mapsto \mathcal{P}ol(t_i) \mid 1 \leq i \leq n\}$ if $\mathcal{P}ol(f)$ is defined (i.e., $f \in \mathcal{F}'$), and otherwise, $\mathcal{P}ol(f(t_1,\ldots,t_n)) = f(\mathcal{P}ol(t_1),\ldots,\mathcal{P}ol(t_n))$. In the following, we use $\mathcal{R}$ as a constrained TRS over $(\mathcal{F},\mathcal{G}_\mathbb{Z},\mathcal{P}_\mathbb{Z},\mathcal{M}_\mathbb{Z})$ without notice. To simplify the presentation, we introduce a weaker version of the PI-based processor in [22].

**Definition 3.2 ([22])** *Let $\mathcal{P}ol$ be a linear PI for $\mathcal{D}^\sharp_\mathcal{R}$*[5] *such that $\mathcal{P}ol(s^\sharp), \mathcal{P}ol(t^\sharp) \in T(\mathcal{G}_\mathbb{Z},\mathcal{V})$ for all $s^\sharp \to t^\sharp$ $[\![ \varphi ]\!] \in \mathcal{S}$,*[6]

**(A1)** *for any $s^\sharp \to t^\sharp$ $[\![ \varphi ]\!] \in \mathcal{S}$, $\mathcal{V}ar(\mathcal{P}ol(t^\sharp)) \subseteq \mathcal{V}ar(\varphi) \cup \mathcal{V}ar(\mathcal{P}ol(s^\sharp))$, and*

**(S1)** *for any $s^\sharp \to t^\sharp$ $[\![ \varphi ]\!] \in \mathcal{S}$, $\varphi \Rightarrow \mathcal{P}ol(s^\sharp) \geq \mathcal{P}ol(t^\sharp)$ is $\mathcal{M}_\mathbb{Z}$-valid.*

*Then, the* PI-based processor $Proc_{\mathsf{PI}}$ *is defined as follows:*

$$Proc_{\mathsf{PI}}(\mathcal{S}) = \{\ \mathcal{S}\setminus\mathcal{S}_>,\ \mathcal{S}\setminus\mathcal{S}_{\mathsf{bound}},\ \mathcal{S}\setminus\mathcal{S}_{\mathsf{filter}}\ \}$$

*where*

- $\mathcal{S}_> = \{s^\sharp \to t^\sharp\ [\![ \varphi ]\!] \in \mathcal{S} \mid \varphi \Rightarrow \mathcal{P}ol(s^\sharp) > \mathcal{P}ol(t^\sharp)$ *is $\mathcal{M}_\mathbb{Z}$-valid*$\}$,
- $\mathcal{S}_{\mathsf{bound}} = \{s^\sharp \to t^\sharp\ [\![ \varphi ]\!] \in \mathcal{S} \mid \varphi \Rightarrow \mathcal{P}ol(s^\sharp) > c_0$ *is $\mathcal{M}_\mathbb{Z}$-valid for some $c_0 \in T(\mathcal{G}_\mathbb{Z})$*$\}$, *and*

[4]In this paper, we do not consider disproving termination, and thus, we do not formalize the case that DP processors return "no" [14].

[5]$\mathcal{P}ol$ is not defined for any symbols in $\mathcal{F}$.

[6]This condition ensures that all the uninterpreted function symbols in $\mathcal{S}$ are dropped by applying $\mathcal{P}ol$ to pairs in $\mathcal{S}$. However, the application of $\mathcal{P}ol$ to an instance of a pair in $\mathcal{S}$ may contain an uninterpreted function symbol.

- $\mathcal{S}_{\mathsf{filter}} = \{s^\sharp \to t^\sharp \; [\![\,\varphi\,]\!] \in \mathcal{S} \mid \mathcal{V}ar(\mathcal{P}ol(s^\sharp)) \subseteq \mathcal{V}ar(\varphi)\}$.

To make $\mathcal{S}$ smaller via $Proc_{\mathsf{PI}}$, we need $\mathcal{S}_> \neq \emptyset$, $\mathcal{S}_{\mathsf{bound}} \neq \emptyset$, and $\mathcal{S}_{\mathsf{filter}} \neq \emptyset$. The idea of the PI-based processor in Definition 3.2 is that an infinite $\mathcal{S}$-chain which contains each pair in $\mathcal{S}_> \cup \mathcal{S}_{\mathsf{bound}} \cup \mathcal{S}_{\mathsf{filter}}$ infinitely many times can be transformed into an infinite bounded strictly-decreasing sequence of integers, i.e., all the elements in the sequence is greater than or equal to a lower bound. Pairs in $\mathcal{S}_\geq$ ensures that the sequence is decreasing; Pairs in $\mathcal{S}_>$ ensures that in focusing on them, the sequence is strictly decreasing; Pairs in $\mathcal{S}_{\mathsf{bound}}$ ensures the existence of the lower bound; Pairs in $\mathcal{S}_{\mathsf{filter}}$ ensures the existence of a pair which is reduced by $\mathcal{P}ol$ to an interpreted ground term (i.e., an integer); (**A1**) ensures that all pairs in the sequence appeared after those in $\mathcal{S}_{\mathsf{filter}}$ can be reduced by $\mathcal{P}ol$ to integers.

**Theorem 3.3 ([22])** *$Proc_{\mathsf{PI}}$ is sound and complete.*

*Example 3.4* Consider $\mathcal{R}_1$ and its dependency pairs $DP(\mathcal{R}_1)$ in Section 1 again. Let us try to apply $Proc_{\mathsf{PI}}$ to the DP problem $DP(\mathcal{R}_1)$. Let $\mathcal{P}ol$ be a linear PI such that $\mathcal{P}ol(\mathsf{f}^\sharp) = a_0 + a_1 x_1$. To satisfy the condition "$\mathcal{P}ol(s^\sharp), \mathcal{P}ol(t^\sharp) \in T(\mathcal{G}_\mathbb{Z}, \mathcal{V})$ for all $s^\sharp \to t^\sharp \; [\![\,\varphi\,]\!] \in \mathcal{S}$", $a_1$ has to be 0 since $\mathsf{f} \notin \mathcal{D}^\sharp_\mathcal{R}$. Thus, $\mathcal{P}ol(\mathsf{f}^\sharp) = a_0$, and hence $\mathcal{S}_> = \emptyset$. Therefore, $Proc_{\mathsf{PI}}(DP(\mathcal{R}_1)) = \{\ DP(\mathcal{R}_1)\ \}$ and $Proc_{\mathsf{PI}}$ does not work for the DP problem $DP(\mathcal{R}_1)$. Note that the other DP processors based on *strongly connected components* or *the subterm criterion* (cf. [22]) do not work for this DP problem, either.

## 4 From Dependency Chains to Monotone Sequences of Integers

PIs satisfying the conditions in Definition 3.2 transform $\mathcal{S}$-chains into bounded decreasing sequences of integers. Focusing on such PIs, we obtain the following corollary from Definition 3.2 and Theorem 3.3.

**Corollary 4.1** *Let $\mathcal{P}ol$ be a linear PI for $\mathcal{D}^\sharp_\mathcal{R}$ such that $\mathcal{P}ol(s^\sharp), \mathcal{P}ol(t^\sharp) \in T(\mathcal{G}_\mathbb{Z}, \mathcal{V})$ for all $s^\sharp \to t^\sharp \; [\![\,\varphi\,]\!] \in \mathcal{S}$, and both* (**A1**) *and* (**S1**) *in Definition 3.2 hold. Then, every $\mathcal{S}$-chain $s^\sharp_0\sigma_0 \to_{\varepsilon,\mathcal{S}} t^\sharp_0\sigma_0 \to^*_{>\varepsilon,\mathcal{R}} s^\sharp_1\sigma_1 \to_{\varepsilon,\mathcal{S}} t^\sharp_1\sigma_1 \to^*_{>\varepsilon,\mathcal{R}} \cdots$ starting with $s^\sharp_0 \to t^\sharp_0 \; [\![\,\varphi_0\,]\!]$ satisfying $\mathcal{V}ar(\mathcal{P}ol(s^\sharp_0)) \subseteq \mathcal{V}ar(\varphi_0)$ can be transformed into a decreasing sequence $\mathcal{P}ol(s^\sharp_0\sigma_0) \geq \mathcal{P}ol(t^\sharp_0\sigma_0) \geq \mathcal{P}ol(s^\sharp_1\sigma_1) \geq \mathcal{P}ol(t^\sharp_1\sigma_1) \geq \cdots$ of integers such that*

- *$>$ appears infinitely many times if $s^\sharp \to t^\sharp \; [\![\,\varphi\,]\!] \in \mathcal{S}_>$ in Definition 3.2 appears in the $\mathcal{S}$-chain infinitely many times, and*
- *the sequence is bounded (i.e., there exists an integer $n$ such that $\mathcal{P}ol(s^\sharp_i) \geq n$ for all $i$) if $s^\sharp \to t^\sharp \; [\![\,\varphi\,]\!] \in \mathcal{S}_{\mathsf{bound}}$ in Definition 3.2 appears in the $\mathcal{S}$-chain infinitely many times.*

In this section, we show sufficient conditions of a linear PI for transforming dependency chains into monotone sequences of integers, strengthening the PI-based processor $Proc_{\mathsf{PI}}$. The difference from $Proc_{\mathsf{PI}}$ is to take $\mathcal{R}$ into account.

### 4.1 The Existing Approach to Transformation of Chains into Decreasing Sequences

As the first step, we follow the existing approach in [5]. To this end, we recall the notion of *reducible positions* [5]. A natural number $i$ is a *reducible position* of a marked symbol $f^\sharp$ w.r.t. $\mathcal{S}$ if there is a dependency pair $s^\sharp \to f^\sharp(t_1, \ldots, t_n) \; [\![\,\varphi\,]\!] \in \mathcal{S}$ such that $t_i \notin T(\mathcal{G}, \mathcal{V}ar(\varphi))$.[7]

To extract rewrite sequences of $\mathcal{R}$ in transforming chains into sequences of integers, for $f/n$ and $\mathcal{P}ol(f^\sharp) = a_0 + a_1 x_1 + \cdots + a_n x_n$, $Proc_{\mathsf{PI}}$ requires the coefficient $a_i$ of any reducible position $i$ of $f^\sharp$ w.r.t. $\mathcal{S}$ to be 0. Due to this requirement, in applying $Proc_{\mathsf{PI}}$, we do not have to take into account rules in $\mathcal{R}$. However, as seen in Example 3.4, this requirement makes $Proc_{\mathsf{PI}}$ ineffective in the case where all

[7] In [5], $t_i \notin T(\mathcal{G}, \mathcal{V})$ is required but in this paper, we require a stronger condition "$t_i \notin T(\mathcal{G}, \mathcal{V}ar(\varphi))$" that is more essential for this notion.

arguments of marked symbols are reducible positions. For this reason, we relax this requirement as in [5] by making a linear PI $\mathcal{P}ol$ for $\mathcal{D}^\sharp_\mathcal{R} \cup \mathcal{F}$ satisfy the following conditions:

(**A2**) Any reduction of $\mathcal{R}$ for uninterpreted symbols in $\mathcal{F}$ does not happen in the second argument of the subtraction operator—for any $u \to v$ $[\![\varphi]\!] \in \mathcal{R} \cup \mathcal{S}$ and any subterm $v'$ of $v$, if $v'$ is rooted by the subtraction symbol "$-$", then $v'|_2 \in T(\mathcal{G}_\mathbb{Z}, \mathcal{V}ar(\varphi))$;

(**A3**) $b_i \geq 0$ for all $1 \leq i \leq n$ and for any $f/n \in \mathcal{F}$ with $\mathcal{P}ol(f) = b_0 + b_1x_1 + \cdots + b_nx_n$;

(**R1**) $\varphi \Rightarrow \mathcal{P}ol(\ell) \geq \mathcal{P}ol(r)$ is $\mathcal{M}_\mathbb{Z}$-valid for any $\ell \to r$ $[\![\varphi]\!] \in \mathcal{R}$;

(**P1**) $a_i \geq 0$ for any reducible position $i$ of $f^\sharp$ and for any $f/n \in \mathcal{D}_\mathcal{R}$ with $\mathcal{P}ol(f^\sharp) = a_0 + a_1x_1 + \cdots + a_nx_n$.

The first three conditions ensure that for any term $s, t \in T(\mathcal{F} \cup \mathcal{G})$, if $s \to_\mathcal{R} t$, then $\mathcal{P}ol(s) \geq \mathcal{P}ol(t)$. The first and last conditions ensure that for any term $s, t \in T(\mathcal{F} \cup \mathcal{G})$ with $root(s) \in \mathcal{D}_\mathcal{R}$, if $s^\sharp \to_\mathcal{R} t^\sharp$, then $\mathcal{P}ol(s^\sharp) \geq \mathcal{P}ol(t^\sharp)$. Note that (**A1**) and $\mathcal{S}_\mathsf{filter}$ are no longer required because $\mathcal{P}ol$ interprets all uninterpreted symbols.

*Example 4.2* Let $\mathcal{P}ol$ be a linear PI such that $\mathcal{P}ol(\mathsf{f}^\sharp) = a_0 + a_1x_1$ and $\mathcal{P}ol(\mathsf{f}) = b_0 + b_1x_1$ with $a_1 \geq 0$ and $b_1 \geq 0$. To transform $DP(\mathcal{R}_1)$-chains into decreasing sequences of integers, both $\mathsf{s}^{101}(\mathsf{0}) > x \Rightarrow \mathcal{P}ol(\mathsf{f}(x)) \geq \mathcal{P}ol(\mathsf{f}(\mathsf{f}(\mathsf{s}^{11}(x))))$ (i.e., $101 > x \Rightarrow b_0 + b_1x \geq b_0 + b_1(b_0 + b_1(x + 11)))$ and $\neg(\mathsf{s}^{101}(\mathsf{0}) > x) \Rightarrow \mathcal{P}ol(\mathsf{f}(x)) \geq \mathcal{P}ol(\mathsf{p}^{10}(x))$ (i.e., $101 \leq x \Rightarrow b_0 + b_1x \geq x - 10$) have to be $\mathcal{M}_\mathsf{LIA}$-valid. However, there is no assignment for $a_0, a_1, b_0, b_1$ ensuing the validity of the two formuas.

## 4.2 Transforming Rewrite Sequences into Increasing Sequences of Integers

As seen in Example 4.2, for $\mathcal{R}_1$, it is impossible for any linear PI to ensure (**R1**). To transform dependency chains into decreasing sequences, the coefficient for a reducible position (i.e., $a_i$ of $\mathcal{P}ol(f^\sharp) = a_0 + a_1x_1 + \cdots + a_nx_n$ with reducible position $i$) has to be a non-negative integer because any rewrite sequences appears below the reducible position is transformed into a decreasing sequence. However, for that purpose, all coefficients for reducible positions may be negative if rewrite sequences are transformed into increasing sequences, i.e., we require that

(**R2**) $\varphi \Rightarrow \mathcal{P}ol(\ell) \leq \mathcal{P}ol(r)$ is $\mathcal{M}_\mathbb{Z}$-valid for any $\ell \to r$ $[\![\varphi]\!] \in \mathcal{R}$, and

(**P2**) $a_i \leq 0$ for any reducible position $i$ of $f^\sharp$ and for any $f/n \in \mathcal{D}_\mathcal{R}$ with $\mathcal{P}ol(f^\sharp) = a_0 + a_1x_1 + \cdots + a_nx_n$.

*Example 4.3* Let $\mathcal{P}ol$ be a linear PI such that $\mathcal{P}ol(\mathsf{f}^\sharp) = 209 - 2x_1$ and $\mathcal{P}ol(\mathsf{f}) = -10 + x_1$. Then, all (**A2**), (**A3**), (**R2**), and (**P1**) are satisfied, and $\mathcal{S}_> = \mathcal{S}_\mathsf{bound} = \{\ (3),\ (4)\ \}$. This means that every $DP(\mathcal{R}_1)$-chain can be transformed into a decreasing sequence of integers such that

- $>$ appears infinitely many times if (3) or (4) appears in the chain infinitely many times, and
- the sequence is bounded if (3) or (4) appears in the chain infinitely many times.

Therefore, there is no infinite $DP(\mathcal{R}_1)$-chain, and hence $\mathcal{R}_1$ is terminating.

## 4.3 Transforming Dependency Chains into Increasing Sequences of Integers

The role of PI $\mathcal{P}ol$ in $Proc_\mathsf{PI}$ is to transform dependency chains into bounded *decreasing* sequences of integers, and to drop a dependency pair $s^\sharp \to t^\sharp$ $[\![\varphi]\!] \in \mathcal{S}$ such that $\varphi \Rightarrow \mathcal{P}ol(s^\sharp) > \mathcal{P}ol(t^\sharp)$ is $\mathcal{M}_\mathbb{Z}$-valid. Since transformed sequences are bounded, the sequences do not have to be decreasing, i.e., they may be bounded increasing sequences. To transform dependency chains into increasing sequences, we invert $\geq$ in (**S1**) and $>$ of $\mathcal{S}_>$ as follows:

- $\mathcal{S}_> = \{s^\sharp \to t^\sharp\ [\![\varphi]\!] \in \mathcal{S} \mid \varphi \Rightarrow \mathcal{P}ol(s^\sharp) < \mathcal{P}ol(t^\sharp)$ is $\mathcal{M}_\mathbb{Z}$-valid$\}$, and

(**S2**) for any $s^\sharp \to t^\sharp$ $[\![\varphi]\!] \in \mathcal{S}$, $\varphi \Rightarrow \mathcal{P}ol(s^\sharp) \leq \mathcal{P}ol(t^\sharp)$ is valid.

For rewrite sequences, we have two ways to transform them (into either decreasing or increasing sequences) and thus, we have the following two combinations to transform dependency chains into increasing sequences.

- When we transform rewrite sequences into decreasing sequences as in Section 4.1, we require (**A2**), (**A3**), (**R1**), (**P2**), and (**S2**).
- When we transform rewrite sequences into increasing sequences as in Section 4.2, we require (**A2**), (**A3**), (**R2**), (**P1**), and (**S2**).

For the both cases above, to ensure the existence of an upper bound, we need a dependency pair $s^\sharp \to t^\sharp\ [\![\,\varphi\,]\!] \in \mathcal{S}$ such that $\varphi \Rightarrow \mathcal{P}ol(s^\sharp) \leq c_0$ is $\mathcal{M}_\mathbb{Z}$-valid for some $c_0 \in T(\mathcal{G}_\mathbb{Z})$.

## 4.4 Improving the PI-based Processor

Finally, we formalize the ideas in previous sections as an improvement of the PI-based processor $Proc_{\mathsf{PI}}$.

**Definition 4.4** *Suppose that* (**A2**) *for any* $u \to v\ [\![\,\varphi\,]\!] \in \mathcal{R} \cup \mathcal{S}$ *and any subterm* $v'$ *of* $v$*, if* $v'$ *is rooted by the subtraction symbol "$-$", then* $v'|_2 \in T(\mathcal{G}_\mathbb{Z}, \mathcal{V}ar(\varphi))$*. Let* $(\bowtie_1, \bowtie_2, \bowtie_3) \in \{(>,\geq,\geq),(<,\geq,\leq),(>,\leq,\leq),(<,\leq,\geq)\}$*, and* $\mathcal{P}ol$ *a linear PI for* $\mathcal{D}^\sharp_\mathcal{R} \cup \mathcal{F}$ *such that*

- $b_i \geq 0$ *for all* $1 \leq i \leq n$ *and for any* $f/n \in \mathcal{F}$ *with* $\mathcal{P}ol(f) = b_0 + b_1x_1 + \cdots + b_nx_n$*,*
- $\varphi \Rightarrow \mathcal{P}ol(\ell) \bowtie_2 \mathcal{P}ol(r)$ *is* $\mathcal{M}_\mathbb{Z}$*-valid for any* $\ell \to r\ [\![\,\varphi\,]\!] \in \mathcal{R}$*,*
- $a_i \bowtie_3 0$ *for any reducible position* $i$ *of* $f^\sharp$ *and for any* $f/n \in \mathcal{D}_\mathcal{R}$ *with* $\mathcal{P}ol(f^\sharp) = a_0 + a_1x_1 + \cdots + a_nx_n$*, and*
- $\varphi \Rightarrow (\mathcal{P}ol(s^\sharp) \bowtie_1 \mathcal{P}ol(t^\sharp) \vee \mathcal{P}ol(s^\sharp) \simeq \mathcal{P}ol(t^\sharp))$ *is* $\mathcal{M}_\mathbb{Z}$*-valid for any* $s^\sharp \to t^\sharp\ [\![\,\varphi\,]\!] \in \mathcal{S}$*.*

*Then, the* PI-based processor $Proc_{(\bowtie_1,\bowtie_2,\bowtie_3)}$ *is defined as follows:*

$$Proc_{(\bowtie_1,\bowtie_2,\bowtie_3)}(\mathcal{S}) = \{\ \mathcal{S} \setminus \mathcal{S}_\bowtie,\ \mathcal{S} \setminus \mathcal{S}_{\mathsf{bound}}\ \}$$

*where*

- $\mathcal{S}_\bowtie = \{s^\sharp \to t^\sharp\ [\![\,\varphi\,]\!] \in \mathcal{S} \mid \varphi \Rightarrow \mathcal{P}ol(s^\sharp) \bowtie_1 \mathcal{P}ol(t^\sharp)$ *is* $\mathcal{M}_\mathbb{Z}$*-valid*$\}$*, and*
- $\mathcal{S}_{\mathsf{bound}} = \{s^\sharp \to t^\sharp\ [\![\,\varphi\,]\!] \in \mathcal{S} \mid \varphi \Rightarrow \mathcal{P}ol(s^\sharp) \bowtie_1 c_0$ *is* $\mathcal{M}_\mathbb{Z}$*-valid for some* $c_0 \in T(\mathcal{G}_\mathbb{Z})\,\}$*.*

**Theorem 4.5** $Proc_{(>,\geq,\geq)}$*,* $Proc_{(<,\geq,\leq)}$*,* $Proc_{(>,\leq,\leq)}$*, and* $Proc_{(<,\leq,\geq)}$ *are sound and complete.*

By definition, it is clear that $Proc_{(>,\geq,\geq)}$ and $Proc_{(<,\geq,\leq)}$ are the same functions from theoretical point of view, and $Proc_{(>,\leq,\leq)}$ and $Proc_{(<,\leq,\geq)}$ are so. For example, given a DP problem $\mathcal{S}$ and a linear PI $\mathcal{P}ol_1$ satisfying the conditions of $Proc_{(>,\geq,\geq)}$, we can construct a linear PI $\mathcal{P}ol_2$ such that $\mathcal{P}ol_2$ satisfies the conditions of $Proc_{(<,\geq,\leq)}$ and $Proc_{(>,\geq,\geq)}(\mathcal{S}) = Proc_{(<,\geq,\leq)}(\mathcal{S})$.

## 4.5 Implementation and Experiments

We implemented the new PI-based processors $Proc_{(>,\geq,\geq)}$, $Proc_{(<,\geq,\leq)}$, $Proc_{(>,\leq,\leq)}$, and $Proc_{(<,\leq,\geq)}$ in Cter, a termination prover based on the techniques in [22]. Those processors first generate a template of a linear PI such as $\mathcal{P}ol(f) = a_0 + a_1x_1 + \cdots a_nx_n$ with non-fixed coefficients $a_0, a_1, \ldots, a_n$, producing a non-linear integer arithmetic formula that belongs to `NIA`, a logic category of SMT-LIB[8]. Satisfiability of the generated formula corresponds to the existence of the PI satisfying the conditions that the processors require. Then, the processors call Z3 [20], an SMT solver, to find an expected PI—if Z3 returns "`unsat`", then there exists no PI satisfying the conditions.

Table 1 illustrates the results of experiments to prove termination of $\mathcal{R}_1$ using one of the new processors. In the third case, $Proc_{(>,\leq,\leq)}$ was applied twice—it first decomposes the initial problem to $\{(3)\}$

[8]`http://smtlib.cs.uiowa.edu`

Table 1: the result of experiments to prove termination of $\mathcal{R}_1$ using the new PI-based processors

| Used processor | Chains | Rewrite sequences | Result | Time (sec.) | Output of Z3 | $\mathcal{P}ol(\mathsf{f})$ | $\mathcal{P}ol(\mathsf{f}^\sharp)$ | $c_0$ |
|---|---|---|---|---|---|---|---|---|
| $Proc_{(>,\geq,\geq)}$ | decreasing | decreasing | fail | 37 | unsat | — | — | – |
| $Proc_{(<,\geq,\leq)}$ | increasing | decreasing | fail | 22 | unsat | — | — | – |
| $Proc_{(>,\leq,\leq)}$ | decreasing | increasing | success | 82 | sat | $-10+x_1$ | $1120-11x_1$ | 21 |
| | | | | | sat | $-10+x_1$ | $200-2x_1$ | 0 |
| $Proc_{(<,\leq,\geq)}$ | increasing | increasing | success | 176 | sat | $-10+x_1$ | $-702+7x_1$ | $-2$ |

and then solves $\{(3)\}$. Experiments are conducted on a machine running Ubuntu 14.04 LTS equipped with an Intel Core i5 CPU at 3.20 GHz with 8 GB RAM. Surprisingly, for each pair of processors with the same power, $(Proc_{(>,\geq,\geq)}, Proc_{(<,\geq,\leq)})$ and $(Proc_{(>,\leq,\leq)}, Proc_{(<,\leq,\geq)})$, the running times are quite different. The reason of these differences might be caused by how Z3 searches assignments that satisfy given formulas. Note that we improved the original PI-based processor in [22]—not the simplified version introduced in this paper—and thus the bounds do not fit Definition 4.4 while they are correct.

## 5 Conclusion

In this paper, we showed sufficient conditions of PIs for transforming dependency chains into bounded monotone sequences of integers, and improved the PI-based processor proposed in [22], providing four PI-based processors. We showed that two of them are useful to prove termination of a constrained TRS defining the McCarthy 91 function over the integers.

The execution time of the proposed processors are larger than we expected, and we would like to improve efficiency. Given a DP problem, the current implementation produces a single large quantified non-linear formula of integer arithmetic expressions, and passes the formula to Z3—Z3 may spend much time to solve such a complicated formula. One of our future work is to improve efficiency of the implementation by simplifying formulas passed to Z3.

One of the important related work is the method in [8]—our PI-based processors would be simplified variants of the method while it has to be extended to constrained rewriting. We will extend the method to our setting and compare our processors with the extended method from theoretical point of view.

**Acknowledgement** We thank the anonymous reviewers for their useful remarks for further development of our PI-based processors.

## References

[1] Beatriz Alarcón, Fabian Emmes, Carsten Fuhs, Jürgen Giesl, Raúl Gutiérrez, Salvador Lucas, Peter Schneider-Kamp & René Thiemann (2008): *Improving Context-Sensitive Dependency Pairs*. In: Proc. LPAR 2008, Lecture Notes in Computer Science 5330, Springer, pp. 636–651, doi:10.1007/978-3-540-89439-1_44.

[2] Thomas Arts & Jürgen Giesl (2000): *Termination of term rewriting using dependency pairs*. Theor. Comput. Sci. 236(1-2), pp. 133–178, doi:10.1016/S0304-3975(99)00207-8.

[3] Franz Baader & Tobias Nipkow (1998): *Term Rewriting and All That*. Cambridge University Press, doi:10.1145/505863.505888.

[4] Adel Bouhoula & Florent Jacquemard (2008): *Automated Induction with Constrained Tree Automata*. In: Proc. IJCAR 2008, Lecture Notes in Computer Science 5195, Springer, pp. 539–554, doi:10.1007/978-3-540-71070-7_44.

[5] Stephan Falke & Deepak Kapur (2008): *Dependency Pairs for Rewriting with Built-In Numbers and Semantic Data Structures*. In: Proc. RTA 2008, Lecture Notes in Computer Science 5117, Springer, pp. 94–109, doi:10.1007/978-3-540-70590-1_7.

[6] Stephan Falke & Deepak Kapur (2009): *A Term Rewriting Approach to the Automated Termination Analysis of Imperative Programs*. In: Proc. CADE 2009, Lecture Notes in Computer Science 5663, Springer, pp. 277–293, doi:10.1007/978-3-642-02959-2_22.

[7] Stephan Falke & Deepak Kapur (2012): *Rewriting Induction + Linear Arithmetic = Decision Procedure*. In: *Proc. IJCAR 2012*, *Lecture Notes in Computer Science* 7364, Springer, pp. 241–255, doi:10.1007/978-3-642-31365-3_20.

[8] Carsten Fuhs, Jürgen Giesl, Aart Middeldorp, Peter Schneider-Kamp, René Thiemann & Harald Zankl (2008): *Maximal Termination*. In: *Proc. RTA 2008*, *Lecture Notes in Computer Science* 5117, Springer, pp. 110–125, doi:10.1007/978-3-540-70590-1_8.

[9] Carsten Fuhs, Jürgen Giesl, Martin Plücker, Peter Schneider-Kamp & Stephan Falke (2009): *Proving Termination of Integer Term Rewriting*. In: *Proc. RTA 2009*, *Lecture Notes in Computer Science* 5595, Springer, pp. 32–47, doi:10.1007/978-3-642-02348-4_3.

[10] Carsten Fuhs, Cynthia Kop & Naoki Nishida (2017): *Verifying Procedural Programs via Constrained Rewriting Induction*. *ACM Trans. Comput. Log.* 18(2), pp. 14:1–14:50, doi:10.1145/3060143.

[11] Yuki Furuichi, Naoki Nishida, Masahiko Sakai, Keiichirou Kusakari & Toshiki Sakabe (2008): *Approach to Procedural-program Verification Based on Implicit Induction of Constrained Term Rewriting Systems*. *IPSJ Trans. Program.* 1(2), pp. 100–121. In Japanese (a translated summary is available from `http://www.trs.css.i.nagoya-u.ac.jp/crisys/`).

[12] Jürgen Giesl (1997): *Termination of Nested and Mutually Recursive Algorithms*. *J. Autom. Reasoning* 19(1), pp. 1–29, doi:10.1023/A:1005797629953.

[13] Jürgen Giesl, René Thiemann & Peter Schneider-Kamp (2005): *The Dependency Pair Framework: Combining Techniques for Automated Termination Proofs*. In: *Proc. LPAR 2004*, *Lecture Notes in Computer Science* 3452, Springer, pp. 301–331, doi:10.1007/978-3-540-32275-7_21.

[14] Jürgen Giesl, René Thiemann, Peter Schneider-Kamp & Stephan Falke (2006): *Mechanizing and Improving Dependency Pairs*. *J. Autom. Reasoning* 37(3), pp. 155–203, doi:10.1007/s10817-006-9057-7.

[15] Cynthia Kop (2013): *Termination of LCTRSs (extended abstract)*. In: *Proc. WST 2013*, pp. 1–5. Available at `http://www.imn.htwk-leipzig.de/WST2013/papers/paper_12.pdf`.

[16] Cynthia Kop & Naoki Nishida (2013): *Term Rewriting with Logical Constraints*. In: *Proc. FroCoS 2013*, *Lecture Notes in Artificial Intelligence* 8152, pp. 343–358, doi:10.1007/978-3-642-40885-4_24.

[17] Cynthia Kop & Naoki Nishida (2014): *Automatic Constrained Rewriting Induction towards Verifying Procedural Programs*. In: *Proc. APLAS 2014*, *Lecture Notes in Computer Science* 8858, pp. 334–353, doi:10.1007/978-3-319-12736-1_18.

[18] Cynthia Kop & Naoki Nishida (2015): *Constrained Term Rewriting tooL*. In: *Proc. LPAR-20*, *Lecture Notes in Computer Science* 9450, pp. 549–557, doi:10.1007/978-3-662-48899-7_38.

[19] Salvador Lucas & José Meseguer (2014): *2D Dependency Pairs for Proving Operational Termination of CTRSs*. In: *Proc. WRLA 2014*, *Lecture Notes in Computer Science* 8663, Springer, pp. 195–212, doi:10.1007/978-3-319-12904-4_11.

[20] Leonardo Mendonça de Moura & Nikolaj Bjørner (2008): *Z3: An Efficient SMT Solver*. In: *Proc. TACAS 2008*, *Lecture Notes in Computer Science* 4963, Springer, pp. 337–340, doi:10.1007/978-3-540-78800-3_24.

[21] Enno Ohlebusch (2002): *Advanced Topics in Term Rewriting*. Springer, doi:10.1007/978-1-4757-3661-8.

[22] Tsubasa Sakata, Naoki Nishida & Toshiki Sakabe (2011): *On Proving Termination of Constrained Term Rewrite Systems by Eliminating Edges from Dependency Graphs*. In: *Proc. WFLP 2011*, *Lecture Notes in Computer Science* 6816, Springer, pp. 138–155, doi:10.1007/978-3-642-22531-4_9.

[23] Tsubasa Sakata, Naoki Nishida, Toshiki Sakabe, Masahiko Sakai & Keiichirou Kusakari (2009): *Rewriting Induction for Constrained Term Rewriting Systems*. *IPSJ Trans. Program.* 2(2), pp. 80–96. In Japanese (a translated summary is available from `http://www.trs.css.i.nagoya-u.ac.jp/crisys/`).

[24] Yoshihito Toyama (1987): *Confluent Term Rewriting Systems with Membership Conditions*. In: *Proc. CTRS 1987*, *Lecture Notes in Computer Science* 308, Springer, pp. 228–241, doi:10.1007/3-540-19242-5_17.

[25] Hans Zantema (2003): *Termination*. In: *Term Rewriting Systems*, chapter 6, *Cambridge Tracts in Theoretical Computer Science* 55, Cambridge University Press, pp. 181–259, doi:t10.1017/S1471068405222445.