

Dependency Pairs for Rewriting with Built-in Numbers and Semantic Data Structures^{***}

Stephan Falke and Deepak Kapur

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

Abstract. This paper defines an expressive class of constrained equational rewrite systems that supports the use of semantic data structures (e.g., sets or multisets) and contains built-in numbers, thus extending our previous work presented at CADE 2007 [6]. These rewrite systems, which are based on normalized rewriting on constructor terms, allow the specification of algorithms in a natural and elegant way. Built-in numbers are helpful for this since numbers are a primitive data type in every programming language. We develop a dependency pair framework for these rewrite systems, resulting in a flexible and powerful method for showing termination that can be automated effectively. Various powerful techniques are developed within this framework, including a subterm criterion and reduction pairs that need to consider only subsets of the rules and equations. It is well-known from the dependency pair framework for ordinary rewriting that these techniques are often crucial for a successful automatic termination proof. Termination of a large collection of examples can be established using the presented techniques.

1 Introduction

Term rewriting provides a powerful framework for specifying algorithms in the form of rewrite systems that operate on data structures generated using constructors. Many algorithms operate on semantic data structures like finite sets, multisets, or sorted lists (e.g., using Java's collection classes or the OCaml extension Moca [3]). Constructors used to generate such data structures satisfy certain properties, i.e., they are not free. For example, finite sets can be generated using the empty set, singleton sets, and set union. Set union is associative (A), commutative (C), idempotent (I), and has the empty set as unit element (U). Such semantic data structures can be modeled using equational axioms. For sorted lists of numbers we also need to use arithmetic constraints on numbers for specifying relations on constructors.

Building upon our earlier work [6], this paper introduces constrained equational rewrite systems which have three components: (i) \mathcal{R} , a set of constrained rewrite rules operating on semantic data structures, (ii) \mathcal{S} , a set of constrained rewrite rules on constructors, and (iii) \mathcal{E} , a set of equations on constructors.

* Partially supported by NSF grant CCF-0541315.

** *Proceedings of the 19th Conference on Rewriting Techniques and Applications (RTA '08)*, Hagenberg, Austria. Lecture Notes in Computer Science, Springer-Verlag, 2008.

Here, (ii) and (iii) are used for modeling semantic data structures where normalization with \mathcal{S} yields normal forms that are unique up to equivalence w.r.t. \mathcal{E} . The constraints for \mathcal{R} and \mathcal{S} are quantifier-free formulas from Presburger arithmetic. Rewriting in a constrained equational rewrite system is done using a combination of normalized rewriting [19] with validity checking of instantiated constraints. Notice that the OCaml extension *Moca* [3] uses the same strategy for semantic data structures. For a further generalization where the rules from \mathcal{R} are allowed to contain conditions in addition to constraints we refer to [7].

Example 1. This example shows a mergesort algorithm that takes a set and returns a sorted list of the elements of the set. For this, sets are constructed using \emptyset , $\langle \cdot \rangle$ (a singleton set) and \cup , where we use the following sets \mathcal{S} and \mathcal{E} .

$$\begin{aligned}\mathcal{E} &= \{ x \cup (y \cup z) \approx (x \cup y) \cup z, \quad x \cup y \approx y \cup x \} \\ \mathcal{S} &= \{ x \cup \emptyset \rightarrow x, \quad x \cup x \rightarrow x \}\end{aligned}$$

The mergesort algorithm is given by the following constrained rewrite rules.

$$\begin{aligned}\text{merge}(\text{nil}, y) &\rightarrow y & \text{merge}(x, \text{nil}) &\rightarrow x \\ \text{merge}(\text{cons}(x, xs), \text{cons}(y, ys)) &\rightarrow \text{cons}(y, \text{merge}(\text{cons}(x, xs), ys)) & \llbracket x > y \rrbracket \\ \text{merge}(\text{cons}(x, xs), \text{cons}(y, ys)) &\rightarrow \text{cons}(x, \text{merge}(xs, \text{cons}(y, ys))) & \llbracket x \not> y \rrbracket \\ \text{msort}(\emptyset) &\rightarrow \text{nil} & \text{msort}(\langle x \rangle) &\rightarrow \text{cons}(x, \text{nil}) \\ \text{msort}(x \cup y) &\rightarrow \text{merge}(\text{msort}(x), \text{msort}(y))\end{aligned}$$

If rewriting modulo $\mathcal{E} \cup \mathcal{S}$ (or $\mathcal{E} \cup \mathcal{S}$ -extended rewriting) is used with these constrained rewrite rules, then the resulting rewrite relation does not terminate since $\text{msort}(\emptyset) \sim_{\mathcal{E} \cup \mathcal{S}} \text{msort}(\emptyset \cup \emptyset) \rightarrow_{\mathcal{R}} \text{merge}(\text{msort}(\emptyset), \text{msort}(\emptyset))$. \diamond

An important property of constrained equational rewrite systems is termination. While automated termination methods work well for establishing termination of rewrite systems defined on free data structures, they do not easily extend to semantic data structures. Dependency pair methods for showing termination of AC-rewrite systems have been developed [17, 20], and [9] generalized the dependency pair method to equational rewriting under the restriction that the equations are collapse-free (thus disallowing idempotency and unit elements), regular (i.e., the same variables occur on both sides), and linear.

In this paper, we extend the dependency pair framework [11] to constrained equational rewrite systems and present various termination techniques within this framework. This paper is a significant improvement over [6] in two directions:

1. The rewrite relation is a strict generalization of the rewrite relation considered in [6] since numbers are built-in. The resulting class of rewrite systems is highly expressive since domain-specific knowledge about numbers is available. This is helpful since most functional and imperative programming languages include numbers as a primitive data type. This paper only allows natural numbers, but an extension to integers is currently being developed.
2. Even if restricted to the rewrite relation considered in [6], the termination techniques presented in this paper strictly generalize the corresponding techniques presented in our earlier work since we present the following termination techniques that are not yet available in [6]:

- Section 4.2 presents a subterm criterion in the spirit of [15], which is a relatively simple yet surprisingly powerful termination technique.
- In Section 4.3 we show that a technique based on reduction pairs has to consider only subsets of \mathcal{R} , \mathcal{S} , and \mathcal{E} as determined by the dependencies between function symbols. It is well-known from ordinary rewriting [13, 15] that this is often crucial for automatic termination proofs.
- Section 4.4 shows that polynomial interpretations with negative coefficients are applicable in this setting. We can thus also argue about termination due to a bounded increase, which is so far only possible for innermost termination of ordinary rewriting [14].

This paper is organized as follows. In Section 2, the rewrite relation is defined. In Section 3, we present a characterization of termination of constrained equational rewrite systems that is based on dependency pairs, and we extend the dependency pair framework to constrained equational rewriting. Section 4 discusses various termination techniques within this framework, including the improvements discussed above. The proofs omitted from this paper can be found in the full version [5], which also contains a large collection of examples.

2 Normalized Equational Rewriting with Constraints

We assume familiarity with the concepts and notations of term rewriting [2]. We consider terms over two sorts, **nat** and **univ**, and we assume an initial signature $\mathcal{F}_{\mathcal{PA}} = \{0, 1, +\}$ with sorts $0, 1 : \mathbf{nat}$ and $+ : \mathbf{nat} \times \mathbf{nat} \rightarrow \mathbf{nat}$. Properties of natural numbers are modelled using the set $\mathcal{PA} = \{x+(y+z) \approx (x+y)+z, x+y \approx y+x, x+0 \approx x\}$ of equations. For each $k \in \mathbb{N} - \{0\}$, we denote the term $1 + \dots + 1$ (with k occurrences of 1) by \mathbf{k} .

We then extend $\mathcal{F}_{\mathcal{PA}}$ by a finite sorted signature \mathcal{F} . We omit stating the sorts explicitly in examples if they can be inferred. In the following we assume that all terms, contexts, context replacements, substitutions, rewrite rules, equations, etc. are sort correct. For any syntactic construct c we let $\mathcal{V}(c)$ denote the set of variables occurring in c . Similarly, $\mathcal{F}(c)$ denotes the function symbols occurring in c . The root symbol of a term s is denoted by $\text{root}(s)$. The root position of a term is denoted by λ . For an arbitrary set \mathcal{E} of equations and terms s, t we write $s \rightarrow_{\mathcal{E}} t$ iff there exist an equation $u \approx v \in \mathcal{E}$, a substitution σ , and a position $p \in \text{Pos}(s)$ such that $s|_p = u\sigma$ and $t = s[v\sigma]_p$. The symmetric closure of $\rightarrow_{\mathcal{E}}$ is denoted by $\vdash_{\mathcal{E}}$, and the reflexive transitive closure of $\vdash_{\mathcal{E}}$ is denoted by $\sim_{\mathcal{E}}$. For two terms s, t we write $s \sim_{\mathcal{E}}^{\lambda} t$ iff $s = f(s_1, \dots, s_n)$ and $t = f(t_1, \dots, t_n)$ such that $s_i \sim_{\mathcal{E}} t_i$ for all $1 \leq i \leq n$, i.e., if equations are only applied below the root.

An *atomic \mathcal{PA} -constraint* has the form \top (truth), $s \simeq t$ (equality) or $s > t$ (greater) for terms $s, t \in \mathcal{T}(\mathcal{F}_{\mathcal{PA}}, \mathcal{V})$. The set of *\mathcal{PA} -constraints* is defined to be the closure of the set of atomic \mathcal{PA} -constraints under \neg (negation) and \wedge (conjunction). Validity (the constraint is true for all assignments) and satisfiability (the constraint is true for some assignment) of \mathcal{PA} -constraints are defined as usual, where we take the set of natural numbers as universe of concern. We also speak of \mathcal{PA} -validity and \mathcal{PA} -satisfiability. These properties are decidable [22].

We consider *constrained rewrite rules*, which are ordinary rewrite rules together with a \mathcal{PA} -constraint C , i.e., expressions of the form $l \rightarrow r[[C]]$ for terms $l, r \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}_{\mathcal{PA}}, \mathcal{V})$ and a \mathcal{PA} -constraint C such that $\text{root}(l) \in \mathcal{F}$ and $\mathcal{V}(r) \subseteq \mathcal{V}(l)$. In a rule $l \rightarrow r[[\top]]$ the constraint \top is omitted. For a set \mathcal{R} of constrained rewrite rules, the set of *defined symbols* is given by $\mathcal{D}(\mathcal{R}) = \{f \mid f = \text{root}(l) \text{ for some } l \rightarrow r[[C]] \in \mathcal{R}\}$. The set of *constructors* is $\mathcal{C}(\mathcal{R}) = \mathcal{F} - \mathcal{D}(\mathcal{R})$. Notice that according to this definition, the symbols from $\mathcal{F}_{\mathcal{PA}}$ are considered to be neither defined symbols nor constructors.

Properties of non-free data structures are modeled using constructor equations and constrained constructor rules. A *constructor equation* has the form $u \approx v$ with $u, v \in \mathcal{T}(\mathcal{C}(\mathcal{R}), \mathcal{V})$ such that u and v are linear and $\mathcal{V}(u) = \mathcal{V}(v)$. A *constrained constructor rule* is a constrained rewrite rule $l \rightarrow r[[C]]$ with $l, r \in \mathcal{T}(\mathcal{C}(\mathcal{R}), \mathcal{V})$.

Constructor equations and constrained constructor rules give rise to the following rewrite relation, which is based on extended rewriting [21] and requires that the \mathcal{PA} -constraint of the constrained constructor rule is \mathcal{PA} -valid after being instantiated by the matcher used for rewriting.

Definition 2 (Constructor Rewrite Relation, \mathcal{PA} -based Substitutions).

Let \mathcal{E} be a finite set of constructor equations and let \mathcal{S} be a finite set of constrained constructor rules. Then $s \rightarrow_{\mathcal{PA} \parallel \mathcal{E} \setminus \mathcal{S}} t$ iff there exist a rule $l \rightarrow r[[C]] \in \mathcal{S}$, a position $p \in \text{Pos}(s)$, and a \mathcal{PA} -based substitution σ (i.e., $\sigma(x) \in \mathcal{T}(\mathcal{F}_{\mathcal{PA}}, \mathcal{V})$ for all variables x of sort **nat**) such that

- (i) $s|_p \sim_{\mathcal{E} \cup \mathcal{PA}} l\sigma$, (ii) $C\sigma$ is \mathcal{PA} -valid, and (iii) $t = s[r\sigma]_p$.

The reason for restricting substitutions to be \mathcal{PA} -based is that then \mathcal{PA} -validity of the instantiated \mathcal{PA} -constraint can be decided by a decision procedure for \mathcal{PA} -validity. We write $s \xrightarrow{>\lambda}_{\mathcal{PA} \parallel \mathcal{E} \setminus \mathcal{S}} t$ iff $s \rightarrow_{\mathcal{PA} \parallel \mathcal{E} \setminus \mathcal{S}} t$ at a position $p \neq \lambda$, and $s \xrightarrow{!>\lambda}_{\mathcal{PA} \parallel \mathcal{E} \setminus \mathcal{S}} t$ iff s reduces to t in zero or more $\rightarrow_{\mathcal{PA} \parallel \mathcal{E} \setminus \mathcal{S}}^{>\lambda}$ steps such that t is a normal form w.r.t. $\rightarrow_{\mathcal{PA} \parallel \mathcal{E} \setminus \mathcal{S}}^{>\lambda}$.

We combine constrained rewrite rules, constrained constructor rules, and constructor equations into a *constrained equational system*. These systems are a strict generalization of the equational systems considered in [6] since they allow the use of \mathcal{PA} -constraints. The system given in Example 1 is a constrained equational system.

Definition 3 (Constrained Equational System (CESs)). A constrained equational system (CES) has the form $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ for finite sets \mathcal{R} of constrained rewrite rules, \mathcal{S} of constrained constructor rules, and \mathcal{E} of constructor equations such that

1. \mathcal{S} is right-linear (i.e., r is linear for all $l \rightarrow r[[C]] \in \mathcal{S}$),
2. $\sim_{\mathcal{E} \cup \mathcal{PA}}$ commutes over $\rightarrow_{\mathcal{PA} \parallel \mathcal{E} \setminus \mathcal{S}}$ (i.e., the inclusion $\sim_{\mathcal{E} \cup \mathcal{PA}} \circ \rightarrow_{\mathcal{PA} \parallel \mathcal{E} \setminus \mathcal{S}} \subseteq \rightarrow_{\mathcal{PA} \parallel \mathcal{E} \setminus \mathcal{S}} \circ \sim_{\mathcal{E} \cup \mathcal{PA}}$ is satisfied), and
3. $\rightarrow_{\mathcal{PA} \parallel \mathcal{E} \setminus \mathcal{S}}$ is convergent modulo $\sim_{\mathcal{E} \cup \mathcal{PA}}$ (i.e., $\rightarrow_{\mathcal{PA} \parallel \mathcal{E} \setminus \mathcal{S}}$ is terminating and the inclusion $\leftarrow_{\mathcal{PA} \parallel \mathcal{E} \setminus \mathcal{S}}^* \circ \rightarrow_{\mathcal{PA} \parallel \mathcal{E} \setminus \mathcal{S}}^* \subseteq \rightarrow_{\mathcal{PA} \parallel \mathcal{E} \setminus \mathcal{S}}^* \circ \sim_{\mathcal{E} \cup \mathcal{PA}} \circ \leftarrow_{\mathcal{PA} \parallel \mathcal{E} \setminus \mathcal{S}}^*$ is satisfied).

Here, the commutation property intuitively states that if $s \sim_{\mathcal{E} \cup \mathcal{PA}} s'$ and $s' \rightarrow_{\mathcal{PA} \parallel \mathcal{E} \setminus \mathcal{S}} t'$, then $s \rightarrow_{\mathcal{PA} \parallel \mathcal{E} \setminus \mathcal{S}} t$ for some $t \sim_{\mathcal{E} \cup \mathcal{PA}} t'$. If \mathcal{S} does not already satisfy this property then it can be achieved by adding *extended rules* [21, 9].

Some commonly used data structures and their specifications in our framework are listed below, where $\langle \cdot \rangle$ creates a singleton set or multiset, respectively. This list should not be considered exhaustive, i.e., there are further semantic data structures satisfying the conditions of Definition 3. The rule marked by “(*)” is needed to make $\sim_{\mathcal{E} \cup \mathcal{PA}}$ commute over $\rightarrow_{\mathcal{PA} \parallel \mathcal{E} \setminus \mathcal{S}}$.

	Constructors	\mathcal{E}	\mathcal{S}
Sorted lists	nil, cons		$\text{cons}(x, \text{cons}(y, zs))$ $\rightarrow \text{cons}(y, \text{cons}(x, zs)) \llbracket x > y \rrbracket$
Multisets	\emptyset, ins	$\text{ins}(x, \text{ins}(y, zs))$ $\approx \text{ins}(y, \text{ins}(x, zs))$	
Multisets	$\emptyset, \langle \cdot \rangle, \cup$	$x \cup (y \cup z) \approx (x \cup y) \cup z$ $x \cup y \approx y \cup x$	$x \cup \emptyset \rightarrow x$
Sets	\emptyset, ins	$\text{ins}(x, \text{ins}(y, zs))$ $\approx \text{ins}(y, \text{ins}(x, zs))$	$\text{ins}(x, \text{ins}(x, ys)) \rightarrow \text{ins}(x, ys)$
Sets	$\emptyset, \langle \cdot \rangle, \cup$	$x \cup (y \cup z) \approx (x \cup y) \cup z$ $x \cup y \approx y \cup x$	$x \cup \emptyset \rightarrow x$ $x \cup x \rightarrow x$ $(x \cup x) \cup y \rightarrow x \cup y$ (*)
Sorted sets	\emptyset, ins		$\text{ins}(x, \text{ins}(y, zs))$ $\rightarrow \text{ins}(y, \text{ins}(x, zs)) \llbracket x > y \rrbracket$ $\text{ins}(x, \text{ins}(y, zs))$ $\rightarrow \text{ins}(x, zs) \llbracket x \simeq y \rrbracket$

The rewrite relation corresponding to a CES is an extension of the normalized rewrite relation used in [6], which in turn is based on [19]. Notice that the redex is normalized by $\rightarrow_{\mathcal{PA} \parallel \mathcal{E} \setminus \mathcal{S}}^{\lambda}$ before the matcher σ is considered. Also notice that the restriction to \mathcal{PA} -based substitution enforces a kind of innermost rewriting for function symbols with resulting sort **nat**.

Definition 4 (Rewrite Relation). Let $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ be a CES and let s, t be terms. Then $s \xrightarrow{\mathcal{S}}_{\mathcal{PA} \parallel \mathcal{E} \setminus \mathcal{R}} t$ iff there exist a constrained rewrite rule $l \rightarrow r \llbracket C \rrbracket \in \mathcal{R}$, a position $p \in \text{Pos}(s)$, and a \mathcal{PA} -based substitution σ such that

- (i) $s|_p \xrightarrow{\lambda}_{\mathcal{PA} \parallel \mathcal{E} \setminus \mathcal{S}} \circ \sim_{\mathcal{E} \cup \mathcal{PA}}^{\lambda} l\sigma$, (ii) $C\sigma$ is \mathcal{PA} -valid, and (iii) $t = s[r\sigma]_p$.

Example 5. Continuing Example 1 we illustrate $\xrightarrow{\mathcal{S}}_{\mathcal{PA} \parallel \mathcal{E} \setminus \mathcal{R}}$. Notice that we add the rule $(x \cup x) \cup y \rightarrow x \cup y$ to \mathcal{S} as indicated in the table above. Consider the term $t = \text{msort}(\langle 1 \rangle \cup (\langle 3 \rangle \cup \langle 1 \rangle))$ and the substitution $\sigma = \{x \mapsto \langle 3 \rangle, y \mapsto \langle 1 \rangle\}$. We get $t \xrightarrow{\lambda}_{\mathcal{PA} \parallel \mathcal{E} \setminus \mathcal{S}} \text{msort}(\langle 1 \rangle \cup \langle 3 \rangle) \sim_{\mathcal{E} \cup \mathcal{PA}}^{\lambda} \text{msort}(x \cup y)\sigma$ and thus $t \xrightarrow{\mathcal{S}}_{\mathcal{PA} \parallel \mathcal{E} \setminus \mathcal{R}} \text{merge}(\text{msort}(\langle 3 \rangle), \text{msort}(\langle 1 \rangle))$. Continuing the reduction of this term for two more $\xrightarrow{\mathcal{S}}_{\mathcal{PA} \parallel \mathcal{E} \setminus \mathcal{R}}$ steps yields $\text{merge}(\text{cons}(3, \text{nil}), \text{cons}(1, \text{nil}))$. Using $\sigma = \{x \mapsto 3, xs \mapsto \text{nil}, y \mapsto 1, ys \mapsto \text{nil}\}$ and the first rule for merge this term reduces to $\text{cons}(1, \text{merge}(\text{nil}, \text{cons}(3, \text{nil})))$ because the instantiated constraint $(x > y)\sigma = (3 > 1)$ is \mathcal{PA} -valid. With one further $\xrightarrow{\mathcal{S}}_{\mathcal{PA} \parallel \mathcal{E} \setminus \mathcal{R}}$ step we finally obtain the term $\text{cons}(1, \text{cons}(3, \text{nil}))$. \diamond

3 Dependency Pairs

In the following, we extend the dependency pair method in order to show termination of rewriting with CESs. The definition of a dependency pair is essentially the well-known one [1], with the only difference that the dependency pair inherits the constraint of the rule that it is created from. As usual, we introduce a signature \mathcal{F}^\sharp , containing for each function symbol $f \in \mathcal{D}(\mathcal{R})$ the function symbol f^\sharp with the same arity and sorts as f . For a term $t = f(t_1, \dots, t_n)$ we denote the term $f^\sharp(t_1, \dots, t_n)$ by t^\sharp , and for a non-trivial context $D = f(t_1, \dots, E, \dots, t_n)$ (where E is also a context) the context $f^\sharp(t_1, \dots, E, \dots, t_n)$ is denoted by D^\sharp .

Definition 6 (Dependency Pairs). *Let $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ be a CES. The dependency pairs of \mathcal{R} are $\text{DP}(\mathcal{R}) = \{l^\sharp \rightarrow t^\sharp \llbracket C \rrbracket \mid t \text{ is a subterm of } r \text{ with } \text{root}(t) \in \mathcal{D}(\mathcal{R}) \text{ for some } l \rightarrow r \llbracket C \rrbracket \in \mathcal{R}\}$.*

In order to verify termination we rely on the notion of *chains*. Intuitively, a dependency pair corresponds to a recursive call, and a chain represents a possible sequence of calls in a reduction w.r.t. $\xrightarrow{\mathcal{S}}_{\mathcal{PA} \parallel \mathcal{E} \setminus \mathcal{R}}$. In the following we always assume that different (occurrences of) dependency pairs are variable disjoint, and we consider substitutions whose domain may be infinite. Additionally, we assume that all substitutions have $\mathcal{T}(\mathcal{F} \cup \mathcal{F}_{\mathcal{PA}}, \mathcal{V})$ as codomain.

Definition 7 ((Minimal) $(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -Chains). *Let $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ be a CES and let \mathcal{P} be a set of dependency pairs. A (possibly infinite) sequence of dependency pairs $s_1 \rightarrow t_1 \llbracket C_1 \rrbracket, s_2 \rightarrow t_2 \llbracket C_2 \rrbracket, \dots$ from \mathcal{P} is a $(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chain iff there exists a \mathcal{PA} -based substitution σ such that $t_i \sigma \xrightarrow{\mathcal{S}}_{\mathcal{PA} \parallel \mathcal{E} \setminus \mathcal{R}}^* s_{i+1} \sigma \circ \xrightarrow{\mathcal{S}}_{\mathcal{PA} \parallel \mathcal{E} \setminus \mathcal{S}}^{\lambda} \circ \sim_{\mathcal{E} \cup \mathcal{PA}}^{\lambda}$ $s_{i+1} \sigma$ and the instantiated \mathcal{PA} -constraint $C_i \sigma$ is \mathcal{PA} -valid for all $i \geq 1$. The above $(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chain is minimal iff $t_i \sigma$ does not start an infinite $\xrightarrow{\mathcal{S}}_{\mathcal{PA} \parallel \mathcal{E} \setminus \mathcal{R}}$ -reduction for any $i \geq 1$.*

Here, $\xrightarrow{\mathcal{S}}_{\mathcal{PA} \parallel \mathcal{E} \setminus \mathcal{R}}^*$ corresponds to reductions occurring strictly below the root of $t_i \sigma$ (notice that $\text{root}(t_i) \in \mathcal{F}^\sharp$), and $\xrightarrow{\mathcal{S}}_{\mathcal{PA} \parallel \mathcal{E} \setminus \mathcal{S}}^{\lambda} \circ \sim_{\mathcal{E} \cup \mathcal{PA}}^{\lambda}$ corresponds to normalization and matching before applying $s_{i+1} \rightarrow t_{i+1} \llbracket C_i \rrbracket$ at the root position.

Example 8. This example is a variation of an example in [23], modified to operate on sets and to use built-in natural numbers. Sets are modelled using the constructors \emptyset and ins as in Section 2 and the function nats is defined so that $\text{nats}(x, y)$ returns the set $\{z \mid x \leq z \leq y\}$.

$$\begin{array}{ll} \text{inc}(\emptyset) \rightarrow \emptyset & \text{inc}(\text{ins}(x, ys)) \rightarrow \text{ins}(x + 1, \text{inc}(ys)) \\ \text{nats}(0, 0) \rightarrow \text{ins}(0, \emptyset) & \text{nats}(0, y + 1) \rightarrow \text{ins}(0, \text{nats}(1, y + 1)) \\ \text{nats}(x + 1, 0) \rightarrow \emptyset & \text{nats}(x + 1, y + 1) \rightarrow \text{inc}(\text{nats}(x, y)) \end{array}$$

We get four dependency pairs in $\text{DP}(\mathcal{R})$.

$$\text{inc}^\sharp(\text{ins}(x, ys)) \rightarrow \text{inc}^\sharp(ys) \tag{1}$$

$$\text{nats}^\sharp(0, y + 1) \rightarrow \text{nats}^\sharp(1, y + 1) \tag{2}$$

$$\text{nats}^\sharp(x + 1, y + 1) \rightarrow \text{inc}^\sharp(\text{nats}(x, y)) \tag{3}$$

$$\text{nats}^\sharp(x + 1, y + 1) \rightarrow \text{nats}^\sharp(x, y) \tag{4}$$

Using the fourth dependency pair twice, we can construct the $(\text{DP}(\mathcal{R}), \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chain $\text{nats}^\#(x + 1, y + 1) \rightarrow \text{nats}^\#(x, y)$, $\text{nats}^\#(x' + 1, y' + 1) \rightarrow \text{nats}^\#(x', y')$ by considering the \mathcal{PA} -based substitution $\sigma = \{x \rightarrow 1, y \rightarrow 1, x' \rightarrow 0, y' \rightarrow 0\}$. \diamond

Using chains, we obtain the following characterization of termination. This is the key result of the dependency pair approach. The proof is similar to the case of ordinary rewriting and can be found in the full version of this paper [5].

Theorem 9. *Let $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ be a CES. Then $\xrightarrow{\mathcal{S}}_{\mathcal{PA} \parallel \mathcal{E} \setminus \mathcal{R}}$ is terminating if and only if there are no infinite minimal $(\text{DP}(\mathcal{R}), \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chains.*

In the next section we present a number of techniques for showing absence of infinite chains. In order to show soundness of these techniques independently, and in order to obtain flexibility on the order in which these techniques are applied, we follow the spirit of [11] and use a dependency pair framework for the termination analysis of CESs. This framework operates on *DP problems* $(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E})$, where \mathcal{P} is a finite set of dependency pairs and $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ is a CES. DP problems are transformed using *DP processors*. Here, a DP processor is a function that takes a DP problem as input and returns a finite set of DP problems as output. The DP processor Proc is *sound* iff for all DP problems $(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ with an infinite minimal $(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chain there exists a DP problem $(\mathcal{P}', \mathcal{R}', \mathcal{S}', \mathcal{E}') \in \text{Proc}(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ with an infinite minimal $(\mathcal{P}', \mathcal{R}', \mathcal{S}', \mathcal{E}')$ -chain.

For a termination proof of the CES $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ we now start with the initial DP problem $(\text{DP}(\mathcal{R}), \mathcal{R}, \mathcal{S}, \mathcal{E})$ and recursively apply sound DP processors. If all resulting DP problems have been transformed into the empty set, then termination has been shown.

4 DP Processors

This section introduces various sound DP processors. Section 4.1 introduces the estimated dependency graph, which determines which dependency pairs may potentially follow each other in a chain. Section 4.2 presents a subterm criterion in the spirit of [15]. The DP processor of Section 4.3 uses \mathcal{PA} -reduction pairs in order to remove dependency pairs. It makes use of the dependencies between function symbols and thus needs to consider only subsets of \mathcal{R} , \mathcal{S} , and \mathcal{E} . Finally, Section 4.4 shows how polynomial interpretations with negative coefficients can be used as \mathcal{PA} -reduction pairs. The DP processors presented here are strictly more powerful than the corresponding DP processors presented in our previous work [6]. It is well-known from ordinary rewriting that the refined techniques are often crucial for a successful automatic termination proof [13, 15]. Additional DP processors are presented in [5].

4.1 Estimated Dependency Graphs

The DP processor introduced in this section decomposes a DP problem into several independent DP problems. The processor relies on the notion of *estimated*

dependency graphs in order to determine which dependency pairs may potentially follow each other in a chain. Estimated dependency graphs are also used in the dependency pair method for ordinary rewriting [1].

To estimate whether the dependency pair $s_1 \rightarrow t_1[[C_1]]$ may be followed by the dependency pair $s_2 \rightarrow t_2[[C_2]]$ in a chain, subterms of t_1 which might be reduced by $\xrightarrow{\mathcal{S}}_{\mathcal{PA} \parallel \mathcal{E} \setminus \mathcal{R}}$ are abstracted by a fresh variable. Then it is checked whether the term obtained from t_1 in this way and s_2 are $\mathcal{E} \cup \mathcal{S} \cup \mathcal{PA}$ -unifiable.

Definition 10 (Estimated Dependency Graphs). *Let $(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ be a DP problem. The estimated $(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -dependency graph $\text{EDG}(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ has the dependency pairs in \mathcal{P} as nodes and there is an arc from $s_1 \rightarrow t_1[[C_1]]$ to $s_2 \rightarrow t_2[[C_2]]$ iff $\text{CAP}(t_1)$ and s_2 are $\mathcal{E} \cup \mathcal{S} \cup \mathcal{PA}$ -unifiable with a \mathcal{PA} -based unifier μ such that $s_1\mu$ and $s_2\mu$ are normal forms w.r.t. $\xrightarrow{>^\lambda}_{\mathcal{PA} \parallel \mathcal{E} \setminus \mathcal{S}}$ and $C_1\mu$ and $C_2\mu$ are \mathcal{PA} -valid. Here, CAP is defined by¹*

$$\text{CAP}(x) = \begin{cases} x & \text{if } x \text{ is a variable of sort } \mathbf{nat} \\ y & \text{if } x \text{ is a variable of sort } \mathbf{univ} \end{cases}$$

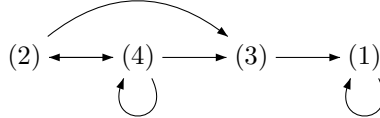
$$\text{CAP}(f(t_1, \dots, t_n)) = \begin{cases} f(\text{CAP}(t_1), \dots, \text{CAP}(t_n)) & \text{if } f \notin \mathcal{D}(\mathcal{R}) \\ y & \text{if } f \in \mathcal{D}(\mathcal{R}) \end{cases}$$

where y is the next variable in an infinite list y_1, y_2, \dots of fresh variables.

It can be shown that this estimate is correct, i.e., if $s_1 \rightarrow t_1[[C_1]]$ may potentially be followed by $s_2 \rightarrow t_2[[C_2]]$ in a $(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chain, then there is a corresponding arc in $\text{EDG}(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E})$. Computing EDG is still hard in general and an implementation could use weaker estimates.

Notice that every infinite $(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chain contains an infinite tail that stays within one *strongly connected component (SCC)* of $\text{EDG}(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E})$, and it is thus sufficient to prove the absence of infinite chains for all SCCs separately.

Example 11. Recall Example 8 and the dependency pairs (1)–(4). We obtain the following estimated dependency graph $\text{EDG}(\text{DP}(\mathcal{R}), \mathcal{R}, \mathcal{S}, \mathcal{E})$.



The graph contains two SCCs and it thus suffices to consider the DP problems $(\{(1)\}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ and $(\{(2), (4)\}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ separately. \diamond

Theorem 12 (DP Processor Based on EDG). *Let Proc be the DP processor with $\text{Proc}(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}) = \{(\mathcal{P}_1, \mathcal{R}, \mathcal{S}, \mathcal{E}), \dots, (\mathcal{P}_n, \mathcal{R}, \mathcal{S}, \mathcal{E})\}$, where $\mathcal{P}_1, \dots, \mathcal{P}_n$ are the SCCs of $\text{EDG}(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E})$.² Then Proc is sound.*

¹ The full version of this paper [5] considers a slightly refined estimation based on the function TCAP to abstract subterms that might be reduced, where TCAP is similar to the function of the same name used for ordinary rewriting in [12].

² Notice, in particular, that $\text{Proc}(\emptyset, \mathcal{R}, \mathcal{S}, \mathcal{E}) = \emptyset$.

4.2 Subterm Criterion

The subterm criterion [15] is a relatively simple technique which is none the less surprisingly powerful. In contrast to the DP processor introduced later in Section 4.3, it only needs to consider the dependency pairs in \mathcal{P} and no rules from \mathcal{R} and \mathcal{S} or equations from \mathcal{E} when operating on a DP problem $(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E})$. The subterm criterion can thus be used to easily handle many DP problems that do not require the more powerful DP processor introduced in Section 4.3.

For ordinary rewriting, the subterm criterion applies a *simple projection* [15] which collapses a term $f^\sharp(t_1, \dots, t_n)$ to one of its direct subterms.

Definition 13 (Simple Projections). *A simple projection is a function π that assigns to every $f^\sharp \in \mathcal{F}^\sharp$ with n arguments a position $i \in \{1, \dots, n\}$. The function that maps $f^\sharp(t_1, \dots, t_n)$ to $t_{\pi(f^\sharp)}$ is also denoted by π .*

For ordinary rewriting the subterm criterion works as follows [15]. If, after applying a simple projection, the right-hand side is a subterm of the left-hand side for all dependency pairs, then all dependency pairs where this subterm relation is strict may be deleted from a DP problem. Crucial for this is the fact that the syntactic subterm relation \triangleright is well-founded.

Notice that the subterm relation modulo $\sim_{\mathcal{E} \cup \mathcal{P}\mathcal{A}}$ is not well-founded since $\mathcal{P}\mathcal{A}$ is collapsing. Thus, the subterm criterion in our framework uses more sophisticated subterm relations, depending on the sorts of the terms. For terms from $\mathcal{T}(\mathcal{F}_{\mathcal{P}\mathcal{A}}, \mathcal{V})$ we use a semantic subterm relation, which also makes use of the constraints that are attached to the dependency pairs.

Definition 14 (Subterms for Sort nat). *Let $s, t \in \mathcal{T}(\mathcal{F}_{\mathcal{P}\mathcal{A}}, \mathcal{V})$ and let C be a $\mathcal{P}\mathcal{A}$ -constraint. Then $s[C] \triangleright_{\text{nat}} t[C]$ iff $C \Rightarrow s > t$ is $\mathcal{P}\mathcal{A}$ -valid and $s[C] \succeq_{\text{nat}} t[C]$ iff $C \Rightarrow s \geq t$ is $\mathcal{P}\mathcal{A}$ -valid.*

For example, $x[x \geq y + 1] \triangleright_{\text{nat}} y[x \geq y + 1]$ since $x \geq y + 1 \Rightarrow x > y$ is $\mathcal{P}\mathcal{A}$ -valid.

If a term s has sort `univ`, then we take the subterm relation modulo $\sim_{\mathcal{E} \cup \mathcal{P}\mathcal{A}}$. Thus, if s, t are terms such that s has sort `univ` and t has sort `univ` or `nat`, then t is a subterm of s iff there exist terms s', t' with $s \sim_{\mathcal{E} \cup \mathcal{P}\mathcal{A}} s'$ and $t' \sim_{\mathcal{E} \cup \mathcal{P}\mathcal{A}} t$ such that t' is a syntactic subterm of s' .

Definition 15 (Subterms for Sort univ). *Let $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ be a CES and let s, t be terms such that s has sort `univ` and t is arbitrary. Then t is a strict subterm of s , written $s \triangleright_{\text{univ}} t$, iff $s \sim_{\mathcal{E} \cup \mathcal{P}\mathcal{A}} \circ \triangleright \circ \sim_{\mathcal{E} \cup \mathcal{P}\mathcal{A}} t$. The term t is a subterm of s , written $s \succeq_{\text{univ}} t$, iff $s \triangleright_{\text{univ}} t$ or $s \sim_{\mathcal{E} \cup \mathcal{P}\mathcal{A}} t$.*

The relation $\triangleright_{\text{univ}}$ is not well-founded in general. Thus, in order to make this relation well-founded, we require \mathcal{E} to be *size preserving*. All data structures given in Section 2 satisfy this property.

Definition 16 (Size Preserving). *Let $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ be a CES. Then \mathcal{E} is size preserving iff $|u| = |v|$ for all $u \approx v \in \mathcal{E}$, where $|t|$ denotes the number of function symbols in the term t .*

Before formally stating a DP processor based on the subterm criterion we illustrate it on an example.

Example 17. Continuing Example 11 we need to handle the two DP problems $(\{(1)\}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ and $(\{(2), (4)\}, \mathcal{R}, \mathcal{S}, \mathcal{E})$. For the first DP problem consider the simple projection $\pi(\text{inc}^\sharp) = 1$. For (1) we get $\pi(\text{inc}^\sharp(\text{ins}(x, ys))) = \text{ins}(x, ys) \triangleright_{\text{univ}} ys = \pi(\text{inc}^\sharp(ys))$ and the only dependency pair can be removed from the DP problem. For the second DP problem, apply the simple projection $\pi(\text{nats}^\sharp) = 2$. Then we have $\pi(\text{nats}^\sharp(0, y + 1)) = y + 1 \succeq_{\text{nat}} y + 1 = \pi(\text{nats}^\sharp(1, y + 1))$ for (2) and $\pi(\text{nats}^\sharp(x + 1, y + 1)) = y + 1 \triangleright_{\text{nat}} y = \pi(\text{nats}^\sharp(x, y))$ for (4). Thus, (4) can be deleted and the resulting DP problem is handled by Theorem 12 since the estimated dependency graph $\text{EDG}(\{(2)\}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ does not contain any SCCs. \diamond

Theorem 18 (DP Processor Based on the Subterm Criterion). *Let π be a simple projection. Then Proc is sound, where $\text{Proc}(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}) =$*

- $\{(\mathcal{P} - \mathcal{P}', \mathcal{R}, \mathcal{S}, \mathcal{E})\}$, if \mathcal{E} is size preserving and $\mathcal{P}' \subseteq \mathcal{P}$ such that
 - $\pi(s) \triangleright_{\text{univ}} \pi(t)$ or $\pi(s)[C] \triangleright_{\text{nat}} \pi(t)[C]$ for all $s \rightarrow t[C] \in \mathcal{P}'$, and
 - $\pi(s) \succeq_{\text{univ}} \pi(t)$ or $\pi(s)[C] \succeq_{\text{nat}} \pi(t)[C]$ for all $s \rightarrow t[C] \in \mathcal{P} - \mathcal{P}'$.
- $\{(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E})\}$, otherwise.

4.3 \mathcal{PA} -reduction Pairs and Function Dependencies

\mathcal{PA} -reduction pairs. The dependency pair framework for ordinary rewriting makes heavy use of reduction pairs (\succsim, \succ) [16]. If all dependency pairs are decreasing w.r.t. $\succsim \cup \succ$, then those decreasing w.r.t. \succ may be deleted from a DP problem. Contrary to Section 4.2, the rewrite rules have to be considered for this as well since (a certain subset of) \mathcal{R} needs to be decreasing w.r.t. \succsim .

In our setting we can relax the requirement that \succsim needs to be monotonic³. We still require \succsim to be \mathcal{F} -monotonic, i.e., $s \succsim t$ implies $D[s] \succsim D[t]$ for all contexts D over $\mathcal{F} \cup \mathcal{F}_{\mathcal{PA}}$ and all terms $s, t \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}_{\mathcal{PA}}, \mathcal{V})$. Monotonicity for contexts with a symbol $f^\sharp \in \mathcal{F}^\sharp$ at its root is only required for positions where a reduction with $\rightarrow_{\mathcal{PA} \parallel \mathcal{E} \setminus \mathcal{S}}$ or $\xrightarrow{\mathcal{S}}_{\mathcal{PA} \parallel \mathcal{E} \setminus \mathcal{R}}$ is possible. A similar observation was already used in [1] for proving innermost termination of ordinary rewriting.

Definition 19 (f^\sharp -monotonic Relations). *Let \bowtie be a relation on terms and let $f^\sharp \in \mathcal{F}^\sharp$. Then \bowtie is f^\sharp -monotonic at position i iff $s \bowtie t$ implies $D^\sharp[s] \bowtie D^\sharp[t]$ for all contexts $D = f(s_1, \dots, s_{i-1}, \square, s_{i+1}, \dots, s_n)$ over $\mathcal{F} \cup \mathcal{F}_{\mathcal{PA}}$ and all terms $s, t \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}_{\mathcal{PA}}, \mathcal{V})$.*

When considering the DP problem $(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E})$, the reduction pair needs to be f^\sharp -monotonic at position i only if \mathcal{P} contains a dependency pair of the form $s \rightarrow f^\sharp(t_1, \dots, t_i, \dots, t_n)[C]$ where $t_i \notin \mathcal{T}(\mathcal{F}_{\mathcal{PA}}, \mathcal{V})$. The reason for this is that no instance of a term from $\mathcal{T}(\mathcal{F}_{\mathcal{PA}}, \mathcal{V})$ can be reduced using $\xrightarrow{\mathcal{S}}_{\mathcal{PA} \parallel \mathcal{E} \setminus \mathcal{R}}$ or $\rightarrow_{\mathcal{PA} \parallel \mathcal{E} \setminus \mathcal{S}}$ in a $(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chain since the substitution used for the chain is \mathcal{PA} -based. It thus suffices to require f^\sharp -monotonicity for all *reducible positions*, which are determined by the DP problem under consideration.

³ A relation \bowtie on terms is *monotonic* iff $s \bowtie t$ implies $D[s] \bowtie D[t]$ for all contexts D .

Definition 20 (Reducible Positions). Let \mathcal{P} be a set of dependency pairs and let $f^\sharp \in \mathcal{F}^\sharp$. Then the set of reducible positions is defined by $\text{RedPos}(f^\sharp, \mathcal{P}) = \{i \mid \text{there exists } s \rightarrow f^\sharp(t_1, \dots, t_i, \dots, t_n) \llbracket C \rrbracket \in \mathcal{P} \text{ such that } t_i \notin \mathcal{T}(\mathcal{F}_{\mathcal{PA}}, \mathcal{V})\}$.

Finally, reduction pairs need to satisfy a property that relates $\vdash_{\mathcal{PA}}$ and $\succeq \cap \succeq^{-1}$.

Definition 21 (\mathcal{PA} -compatible Relations). Let \bowtie be a relation on terms. Then \bowtie is \mathcal{PA} -compatible iff, for all terms $s, t \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}_{\mathcal{PA}}, \mathcal{V})$, $s \vdash_{\mathcal{PA}} t$ implies $D[s] \bowtie \cap \bowtie^{-1} D[t]$ for all contexts D over $\mathcal{F} \cup \mathcal{F}_{\mathcal{PA}}$ and $D^\sharp[s] \bowtie \cap \bowtie^{-1} D^\sharp[t]$ for all contexts $D \neq \square$ over $\mathcal{F} \cup \mathcal{F}_{\mathcal{PA}}$.

Our notion of \mathcal{PA} -reduction pairs depends on the DP problem under consideration. It is similar to the notion of *generalized reduction pairs* [14] in the sense that full monotonicity is not required. However, the generalized reduction pairs of [14] are only applicable in the context of innermost termination of ordinary rewriting.

Definition 22 (\mathcal{PA} -reduction Pairs). Let $(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ be a DP problem and let \succeq and \succ be relations on terms such that \succ is well-founded, \succeq is \mathcal{F} -monotonic and \mathcal{PA} -compatible, and \succeq is f^\sharp -monotonic at position i for all $f^\sharp \in \mathcal{F}^\sharp$ and all $i \in \text{RedPos}(f^\sharp, \mathcal{P})$. Then (\succeq, \succ) is a \mathcal{PA} -reduction pair for $(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ iff \succ is compatible with \succeq , i.e., iff $\succeq \circ \succ \subseteq \succ$ or $\succ \circ \succeq \subseteq \succ$. The relation $\succeq \cap \succeq^{-1}$ is denoted by \sim .

Section 4.4 shows how this definition allows the use of polynomial interpretations with negative coefficients

Notice that we do not require \succeq or \succ to be stable⁴. Indeed, stability for all substitutions is not needed since we only need this property for certain \mathcal{PA} -based substitutions that can be used in $(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chains. These substitutions are indirectly given by the constraints of the dependency pairs and rules that are to be oriented.

Definition 23 (\mathcal{PA} -reduction Pairs on Constrained Terms). Let (\succeq, \succ) be a \mathcal{PA} -reduction pair. Let s, t be terms and let C be a \mathcal{PA} -constraint. Then $s \llbracket C \rrbracket \succeq t \llbracket C \rrbracket$ iff $s\sigma \succeq t\sigma$ for all \mathcal{PA} -based substitutions σ such that $C\sigma$ is \mathcal{PA} -valid. $s \llbracket C \rrbracket \succ t \llbracket C \rrbracket$ is defined analogously.

Function Dependencies. The DP processor based on \mathcal{PA} -reduction pairs makes use of the observation that only certain subsets of \mathcal{R} , \mathcal{S} , and \mathcal{E} need to be considered. The corresponding result for ordinary rewriting is due to [13, 15], where it is also shown that the power of the corresponding DP processor is strictly increased. The main idea is to show that each $(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chain can be transformed into a sequence that only uses subsets $\mathcal{R}' \subseteq \mathcal{R}$, $\mathcal{S}' \subseteq \mathcal{S}$, and $\mathcal{E}' \subseteq \mathcal{E}$. This sequence will not necessarily be a $(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ -chain in our setting,

⁴ A relation \bowtie is *stable* iff $s \bowtie t$ implies $s\sigma \bowtie t\sigma$ for all substitutions σ .

but this is not needed for soundness. A complete proof of the technical result in this section is given in the full version of this paper [5]. The subsets of \mathcal{R} , \mathcal{S} , and \mathcal{E} are based on the dependencies between function symbols. Similar definitions are also used in [13, 15].

Definition 24 (Function Dependencies). *Let $(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ be a DP problem where \mathcal{E} is size preserving. For two function symbols $f, g \in \mathcal{F}$ let $f \sqsupset_{(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E})} g$ iff (i) there exists a rule $l \rightarrow r[C] \in \mathcal{R} \cup \mathcal{S}$ such that $\text{root}(l) = f$ and $g \in \mathcal{F}(r)$, or (ii) there exists an equation $u \approx v$ or $v \approx u$ in \mathcal{E} such that $\text{root}(u) = f$ and $g \in \mathcal{F}(u \approx v)$. Let $\Delta(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}) = \mathcal{F}_{\mathcal{PA}} \cup \{g \mid f \sqsupset_{(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E})}^* g \text{ for some } f \in \mathcal{F} \text{ with resulting sort } \mathbf{nat} \text{ or some } f \in \mathcal{F}(\text{rhs}(\mathcal{P})) - \mathcal{F}^\#\}$, where $\text{rhs}(\mathcal{P})$ denotes the set of right-hand sides of the dependency pairs in \mathcal{P} .*

Subsets $\Delta \subseteq \mathcal{F} \cup \mathcal{F}_{\mathcal{PA}}$ give rise to subsets of \mathcal{R} , \mathcal{S} , and \mathcal{E} in the obvious way.

Definition 25 ($\mathcal{R}(\Delta)$, $\mathcal{S}(\Delta)$, and $\mathcal{E}(\Delta)$). *Let $\Delta \subseteq \mathcal{F} \cup \mathcal{F}_{\mathcal{PA}}$. For $\mathcal{Q} \in \{\mathcal{R}, \mathcal{S}\}$ we define $\mathcal{Q}(\Delta) = \{l \rightarrow r[C] \in \mathcal{Q} \mid \text{root}(l) \in \Delta\}$ and we let $\mathcal{E}(\Delta) = \{u \approx v \in \mathcal{E} \mid \text{root}(u) \in \Delta \text{ or } \text{root}(v) \in \Delta\}$.*

Example 26. Continuing Example 11, recall the DP problems $(\{(1)\}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ and $(\{(2), (4)\}, \mathcal{R}, \mathcal{S}, \mathcal{E})$. These DP problems are hard to handle if all of \mathcal{R} , \mathcal{S} , and \mathcal{E} need to be considered. Using function dependencies we get $\Delta = \mathcal{F}_{\mathcal{PA}}$ for both DP problems. Thus, \mathcal{R} , \mathcal{S} , and \mathcal{E} do not need to be considered at all when handling these DP problems. Using the \mathcal{PA} -reduction pair $(\succsim_{\mathcal{Pol}}, \succ_{\mathcal{Pol}})$ based on a polynomial interpretation with $\mathcal{Pol}(\text{inc}^\#) = x_1$, $\mathcal{Pol}(\text{ins}) = x_2 + 1$, $\mathcal{Pol}(\text{nats}^\#) = x_2$, $\mathcal{Pol}(0) = 0$, $\mathcal{Pol}(1) = 1$ and $\mathcal{Pol}(+) = x_1 + x_2$, the DP problem $(\{(1)\}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ is transformed into the trivial DP problem $(\emptyset, \mathcal{R}, \mathcal{S}, \mathcal{E})$, while the DP problem $(\{(2), (4)\}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ is transformed into $(\{(2)\}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ whose estimated dependency graph does not contain any SCCs. \diamond

The following DP processor is based on \mathcal{PA} -reduction pairs and function dependencies. Again, recall that \mathcal{E} is size preserving for all data structures given in Section 2.

Theorem 27 (DP Processor Based on Function Dependencies). *The DP processor Proc is sound, where $\text{Proc}(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E}) =$*

- $\{(\mathcal{P} - \mathcal{P}', \mathcal{R}, \mathcal{S}, \mathcal{E})\}$, if (\succsim, \succ) is a \mathcal{PA} -reduction pair for $(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E})$, \mathcal{E} is size preserving, $\mathcal{P}' \subseteq \mathcal{P}$, and $\Delta = \Delta(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ such that
 - $s[C] \succ t[C]$ for all $s \rightarrow t[C] \in \mathcal{P}'$,
 - $s[C] \succsim t[C]$ for all $s \rightarrow t[C] \in (\mathcal{P} - \mathcal{P}') \cup \mathcal{R}(\Delta) \cup \mathcal{S}(\Delta) \cup \mathcal{R}_\Pi$, and⁵
 - $u[\top] \sim v[\top]$ for all $u \approx v \in \mathcal{E}(\Delta)$.⁶
- $\{(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E})\}$, otherwise.

If \mathcal{E} is not size preserving a similar DP processor can be used which has to consider all of \mathcal{R} , \mathcal{S} , and \mathcal{E} . In this case, \mathcal{R}_Π does not need to be oriented.

⁵ Here, $\mathcal{R}_\Pi = \{\Pi(x, y) \rightarrow x, \Pi(x, y) \rightarrow y\}$ for a fresh function symbol Π .

⁶ This condition ensures $u\sigma \sim v\sigma$ for all \mathcal{PA} -based substitutions σ .

4.4 Generation of \mathcal{PA} -reduction Pairs

To take advantage of the relaxed requirements on monotonicity and stability and in order to make use of the constraints attached to dependency pairs and rules, we propose to use relations based on polynomial interpretations [18] with polynomials containing coefficients from \mathbb{Z} . A similar kind of polynomial interpretations was used in [14] in the context of innermost termination of ordinary rewriting. The polynomial interpretations with coefficients in \mathbb{Z} used in [15] are also similar but require the use of “max”, which makes reasoning about them more complicated. Furthermore, the approach of [15] requires that rewrite rules are treated like equations, which is very restrictive. Using our approach, rewrite rules do not need to be oriented as equations.

A \mathcal{PA} -polynomial interpretation maps

1. $\mathcal{F}_{\mathcal{PA}}$ to polynomials over \mathbb{N} in the natural way, i.e., $\mathcal{Pol}(0) = 0$, $\mathcal{Pol}(1) = 1$, and $\mathcal{Pol}(+) = x_1 + x_2$,
2. \mathcal{F} to polynomials over \mathbb{N} where $\mathcal{Pol}(f) \in \mathbb{N}[x_1, \dots, x_n]$ if f has arity n , and
3. $\mathcal{F}^\#$ to polynomials over \mathbb{Z} where $\mathcal{Pol}(f^\#) \in \mathbb{Z}[x_1, \dots, x_n]$ if $f^\#$ has arity n .

This mapping is extended to terms by letting $[x]_{\mathcal{Pol}} = x$ for all variables $x \in \mathcal{V}$ and $[f(t_1, \dots, t_n)]_{\mathcal{Pol}} = \mathcal{Pol}(f)([t_1]_{\mathcal{Pol}}, \dots, [t_n]_{\mathcal{Pol}})$ for all $f \in \mathcal{F} \cup \mathcal{F}^\#$.

\mathcal{PA} -polynomial interpretations generate relations on terms in the following way. Here, $\geq^{(\mathbb{Z}, \mathbb{N})}$ and $>^{(\mathbb{Z}, \mathbb{N})}$ mean that the respective relation holds in the integers for all instantiations of the variables by *natural numbers*.

Definition 28 (Relations $\succ_{\mathcal{Pol}}$ and $\succsim_{\mathcal{Pol}}$). *Let \mathcal{Pol} be a \mathcal{PA} -polynomial interpretation. Then $\succ_{\mathcal{Pol}}$ is defined by $s \succ_{\mathcal{Pol}} t$ iff $[s]_{\mathcal{Pol}} \geq^{(\mathbb{Z}, \mathbb{N})} 0$ and $[s]_{\mathcal{Pol}} >^{(\mathbb{Z}, \mathbb{N})} [t]_{\mathcal{Pol}}$. Similarly, $\succsim_{\mathcal{Pol}}$ is defined by $s \succsim_{\mathcal{Pol}} t$ iff $[s]_{\mathcal{Pol}} \geq^{(\mathbb{Z}, \mathbb{N})} [t]_{\mathcal{Pol}}$.*

The relations $\succsim_{\mathcal{Pol}}$ and $\succ_{\mathcal{Pol}}$ give rise to \mathcal{PA} -reduction pairs, where $f^\#$ -monotonicity is translated into conditions on the polynomial $\mathcal{Pol}(f^\#)$.

Theorem 29. *Let $(\mathcal{P}, \mathcal{R}, \mathcal{S}, \mathcal{E})$ be a DP problem and let \mathcal{Pol} be a \mathcal{PA} -polynomial interpretation. Then $(\succsim_{\mathcal{Pol}}, \succ_{\mathcal{Pol}})$ is a \mathcal{PA} -reduction pair if $\mathcal{Pol}(f^\#)$ is weakly increasing in all x_i for which $i \in \text{RedPos}(f^\#, \mathcal{P})$.*

In order to show $s[[C]] \succ_{\mathcal{Pol}} t[[C]]$ it suffices to show that $C \Rightarrow [s]_{\mathcal{Pol}} \geq 0$ and $C \Rightarrow [s]_{\mathcal{Pol}} > [t]_{\mathcal{Pol}}$ are (\mathbb{Z}, \mathbb{N}) -valid, i.e., true in the integers for all instantiations of the variables by natural numbers. While this is undecidable in general it is decidable if all polynomials are linear. Automatic generation of suitable \mathcal{PA} -polynomial interpretations is possible by adapting techniques developed for ordinary polynomial interpretations based on solving non-linear Diophantine constraints [4, 8].

Example 30. One of the leading examples from [14] (obtained from the imperative program `while (x > y) { y = y + 1; }`) can be given more elegantly using built-in numbers. In this example we have $\mathcal{E} = \mathcal{S} = \emptyset$ and there is only a single rewrite rule:

$$\text{eval}(x, y) \rightarrow \text{eval}(x, y + 1) \llbracket x > y \rrbracket$$

The only dependency pair is identical to this rule, but with `eval` replaced by `eval#`. Since $\mathcal{R}(\Delta) = \emptyset$ we do not need consider the rewrite rule when applying the DP processor of Theorem 27, i.e., it suffices to find a \mathcal{PA} -reduction pair (\succsim, \succ) such that $\text{eval}^{\#}(x, y) \llbracket x > y \rrbracket \succ \text{eval}^{\#}(x, y + 1) \llbracket x > y \rrbracket$. For this, consider a \mathcal{PA} -polynomial interpretation with $\text{Pol}(\text{eval}^{\#}) = x_1 - x_2$. We then have $\text{eval}^{\#}(x, y) \llbracket x > y \rrbracket \succ_{\text{Pol}} \text{eval}^{\#}(x, y + 1) \llbracket x > y \rrbracket$ since $x > y \Rightarrow x - y \geq 0$ and $x > y \Rightarrow x - y > x - (y + 1)$ are (\mathbb{Z}, \mathbb{N}) -valid. \diamond

This example demonstrates that built-in numbers are useful for termination proofs of imperative programs operating on numbers since the termination proof is much simpler than the one given in [14]. A large collection of termination proofs for imperative programs is given in the full version of this paper [5].

5 Conclusions and Future Work

We have presented the notion of constrained equational systems for modeling algorithms. Constrained equational systems support semantic data structures and contain built-in numbers. These systems are a strict generalization of the equational systems presented in [6], which do not contain built-in numbers. Since virtually all programming languages have numbers as a primitive data type this extension is helpful for modeling algorithms. We have presented a dependency pair framework for proving termination of constrained equational systems and developed several DP processors within this framework. Most of these DP processors are not available in our previous work [6], while it is well-known from ordinary rewriting that the techniques employed in these DP processors are often crucial for a successful automatic termination proof [13, 15]. The techniques of this paper have been successfully used on a large collection of examples [5].

Termination is only one among several important properties of constrained equational systems. We will study other properties as well, specifically confluence and sufficient completeness. Orthogonal to this we will investigate how the rewrite relation can be generalized by considering other built-in theories, in particular integers instead of natural numbers. The proposed method has not been implemented yet, but we believe that it can be easily implemented within a termination tool like AProVE [10].

References

1. T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *TCS*, 236(1–2):133–178, 2000.
2. F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.
3. F. Blanqui, T. Hardin, and P. Weis. On the implementation of construction functions for non-free concrete data types. In *ESOP '07*, LNCS 4421, pages 95–109, 2007.

4. E. Contejean, C. Marché, A. P. Tomás, and X. Urbain. Mechanically proving termination using polynomial interpretations. *JAR*, 34(4):325–363, 2005.
5. S. Falke and D. Kapur. Dependency pairs for rewriting with built-in numbers and semantic data structures. Technical Report TR-CS-2007-21⁷, Department of Computer Science, University of New Mexico, 2007.
6. S. Falke and D. Kapur. Dependency pairs for rewriting with non-free constructors. In *CADE '07*, LNCS 4603, pages 426–442, 2007.
7. S. Falke and D. Kapur. Operational termination of conditional rewriting with built-in numbers and semantic data structures. Technical Report TR-CS-2007-22⁷, Department of Computer Science, University of New Mexico, 2007.
8. C. Fuhs, J. Giesl, A. Middeldorp, P. Schneider-Kamp, R. Thiemann, and H. Zankl. SAT solving for termination analysis with polynomial interpretations. In *SAT '07*, LNCS 4501, pages 340–354, 2007.
9. J. Giesl and D. Kapur. Dependency pairs for equational rewriting. In *RTA '01*, LNCS 2051, pages 93–108, 2001.
10. J. Giesl, P. Schneider-Kamp, and R. Thiemann. AProVE 1.2: Automatic termination proofs in the dependency pair framework. In *IJCAR '06*, LNCS 4130, pages 281–286, 2006.
11. J. Giesl, R. Thiemann, and P. Schneider-Kamp. The dependency pair framework: Combining techniques for automated termination proofs. In *LPAR '04*, LNCS 3452, pages 301–331, 2005.
12. J. Giesl, R. Thiemann, and P. Schneider-Kamp. Proving and disproving termination of higher-order functions. In *ProCoS '05*, LNCS 3717, pages 216–231, 2005.
13. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Mechanizing and improving dependency pairs. *JAR*, 37(3):155–203, 2006.
14. J. Giesl, R. Thiemann, S. Swiderski, and P. Schneider-Kamp. Proving termination by bounded increase. In *CADE '07*, LNCS 4603, pages 443–459, 2007.
15. N. Hirokawa and A. Middeldorp. Tyrolean termination tool: Techniques and features. *IC*, 205(4):474–511, 2007.
16. K. Kusakari, M. Nakamura, and Y. Toyama. Argument filtering transformation. In *PPDP '99*, LNCS 1702, pages 47–61, 1999.
17. K. Kusakari and Y. Toyama. On proving AC-termination by AC-dependency pairs. *IEICE Transactions on Information and Systems*, E84-D(5):604–612, 2001.
18. D. S. Lankford. On proving term rewriting systems are Noetherian. Memo MTP-3, Mathematics Department, Louisiana Tech University, Ruston, 1979.
19. C. Marché. Normalized rewriting: An alternative to rewriting modulo a set of equations. *JSC*, 21(3):253–288, 1996.
20. C. Marché and X. Urbain. Modular and incremental proofs of AC-termination. *JSC*, 38(1):873–897, 2004.
21. G. E. Peterson and M. E. Stickel. Complete sets of reductions for some equational theories. *J. ACM*, 28(2):233–264, 1981.
22. M. 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.
23. J. Steinbach and U. Kühler. Check your ordering—termination proofs and open problems. Technical Report SR-90-25, Universität Karlsruhe, 1990.

⁷ Available from <http://www.cs.unm.edu/research/tech-reports/>.