Operational Termination of Conditional Rewriting with Built-in Numbers and Semantic Data Structures[⋆†]

Stephan Falke and Deepak Kapur

Department of Computer Science
University of New Mexico
Albuquerque, NM 87131, USA
{spf, kapur}@cs.unm.edu

Abstract. Rewrite systems on free data structures have limited expressive power since semantic data structures like sets or multisets cannot be modeled elegantly. In this work we define a class of rewrite systems that allows the use of semantic data structures. Additionally, built-in natural numbers, including (dis)equality, ordering, and divisibility constraints, are supported. The rewrite mechanism is a combination of normalized equational rewriting with evaluation of conditions and validity checking of instantiated constraints. The framework is highly expressive and allows modeling of algorithms in a natural way.
Termination is one of the most important properties of conditional normalized equational rewriting. For this it is not sufficient to only show well-foundedness of the rewrite relation, but it also has to be ensured that evaluation of the conditions does not loop. The notion of operational termination is a way to capture these properties. In this work we show that it is possible to transform a conditional constrained equational rewrite system into an unconditional one such that termination of the latter implies operational termination of the former. Methods for showing termination of unconditional constrained equational rewrite system are presented in a companion paper.

1 Introduction

Rewrite systems serve as a powerful framework for specifying algorithms in a functional programming style. This is the approach taken in ELAN [10], Maude [2], and theorem provers such as RRL [9], where algorithms are given as terminating rewrite systems that operate on data structures generated by free constructors. Results and powerful automated tools based on term rewriting methods can then be used for analyzing these algorithms.

Many algorithms, however, operate on semantic data structures like finite sets, multisets, or sorted lists. Constructors used to generate such data structures

⋆ Partially supported by NSF grant CCF-0541315.

† This paper is a companion paper to [5]. In order to make the present paper self-contained Sections 1, 2, and 5 have significant overlaps with [5].

satisfy certain properties, i.e., they are not free. For example, finite sets can be generated using the empty set $\emptyset$ and the constructor ins which adds an element to a set. In order to accurately model the semantics of sets we expect that it does not matter in which order elements are added to a set, i.e., we expect that $\mathsf{ins}(x, \mathsf{ins}(y, zs)) \approx \mathsf{ins}(y, \mathsf{ins}(x, zs))$ holds for all elements x, y and all sets z. In order to model the fact that duplicity of elements does not matter we will use arithmetic constraints on numbers in order to specify relations on constructors.

Building upon our earlier work [6], this paper introduces conditional constrained equational rewrite systems which have three components: (i) $\mathcal{R}$, a set of conditional constrained rewrite rules for specifying algorithms 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. Here, (ii) and (iii) are used for modeling semantic data structures. The constraints for $\mathcal{R}$ and $\mathcal{S}$ are Boolean combinations of atomic formulas of the form $s \simeq t$, $s > t$ and $k \mid s$ from Presburger arithmetic. Rewriting in a constrained equational rewrite system is done using a combination of normalized rewriting [12] with evaluation of conditions and validity checking of instantiated constraints. Before rewriting a term with $\mathcal{R}$, the redex is normalized with $\mathcal{S}$, and rewriting is only performed if the instantiated constraint belonging to the rewrite rule from $\mathcal{R}$ is valid and all instantiated conditions for that rewrite rule can be established.

Example 1. This example shows a quicksort algorithm that takes a set and returns a sorted list of the elements of the set. For this, sets are constructed using $\emptyset$ and ins, where ins adds an element to a set. In order to model the semantics of sets we use $\mathcal{S}$ and $\mathcal{E}$ and follows.

$$\begin{aligned} &\mathcal{E}\colon \mathsf{ins}(x, \mathsf{ins}(y, zs)) \approx \mathsf{ins}(y, \mathsf{ins}(x, zs)) \\ &\mathcal{S}\colon \mathsf{ins}(x, \mathsf{ins}(y, zs)) \to \mathsf{ins}(x, zs) \qquad [\![x \simeq y]\!] \end{aligned}$$

Now the quicksort algorithm can be specified by the following conditional constrained rewrite rules.

$$\begin{aligned} \mathsf{app}(\mathsf{nil}, zs) &\to zs \\ \mathsf{app}(\mathsf{cons}(x, ys), zs) &\to \mathsf{cons}(x, \mathsf{app}(ys, zs)) \\ \mathsf{split}(x, \emptyset) &\to \langle \emptyset, \emptyset \rangle \\ \mathsf{split}(x, zs) \to^* \langle zl, zh \rangle \mid \mathsf{split}(x, \mathsf{ins}(y, zs)) &\to \langle \mathsf{ins}(y, zl), zh \rangle \qquad [\![x > y]\!] \\ \mathsf{split}(x, zs) \to^* \langle zl, zh \rangle \mid \mathsf{split}(x, \mathsf{ins}(y, zs)) &\to \langle zl, \mathsf{ins}(y, zh) \rangle \qquad [\![x \not> y]\!] \\ \mathsf{qsort}(\emptyset) &\to \mathsf{nil} \\ \mathsf{split}(x, ys) \to^* \langle yl, yh \rangle \mid \mathsf{qsort}(\mathsf{ins}(x, ys)) &\to \mathsf{app}(\mathsf{qsort}(yl), \mathsf{cons}(x, \mathsf{qsort}(yh))) \end{aligned}$$

Here, $\mathsf{split}(x, ys)$ returns a pair of sets $\langle yl, yh \rangle$ where yl contains all $y \in ys$ such that $x > y$ and yh contains all $y \in ys$ such that $x \not> y$. ◊

Two important properties of conditional constrained equational rewrite systems are termination and the property that evaluation of the conditions does not loop. These properties are captured by the notion of *operational termination* [11, 3]. In this paper we show that operational termination of a conditional constrained equational rewrite system can be reduced to termination of an unconditional constrained equational rewrite system by a simple transformation that takes the order in which the conditions are evaluated into account. This transformation is similar to the transformation used for ordinary conditional rewriting, see, e.g., [13, Definition 7.2.48]. Powerful methods for showing termination of unconditional constrained equational rewrite systems are presented in the companion paper [5]. These methods can thus be used for showing operational termination of conditional constrained equational rewrite systems as well. Using this approach the operational termination of several nontrivial conditional constrained equational rewrite systems is shown in [5].

This paper is organized as follows. In Section 2, the rewrite relation is defined. Since rewrite rules can use arithmetic constraints, we also review constraints in quantifier-free Presburger arithmetic. In Section 3 we formally define the notion of operational termination and show that termination and operational termination coincide for unconditional constrained equational rewrite systems. Section 4 introduces a transformation from conditional constrained equational rewrite systems into unconditional ones. We show that termination of the transformed system implies operational termination of the original system.

2 Conditional Normalized Equational Rewriting with Constraints

We assume familiarity with the concepts and notations of term rewriting [1]. We consider terms over two sorts, `nat` and `univ`, and we assume an initial signature $\mathcal{F}_{\mathcal{PA}} = \{\mathsf{0}, \mathsf{1}, +\}$ with sorts $\mathsf{0}, \mathsf{1} : \mathtt{nat}$ and $+ : \mathtt{nat} \times \mathtt{nat} \to \mathtt{nat}$. Properties of natural numbers are modelled using the set $\mathcal{PA} = \{x+(y+z) \approx (x+y)+z,\ x+y \approx y+x,\ x+\mathsf{0} \approx x\}$ of equations. Due to these properties we occasionally omit parentheses in terms of sort `nat`. For each $k \in \mathbb{N} - \{0\}$, we denote the term $\mathsf{1} + \ldots + \mathsf{1}$ (with k occurrences of $\mathsf{1}$) by k, and for a variable x we let $\mathsf{k}x$ denote the term $x + \ldots + x$ (with k occurrences of x). In the following, $\overline{s}$ denotes the natural number corresponding to the term $s \in \mathcal{T}(\mathcal{F}_{\mathcal{PA}})$.

We then extend $\mathcal{F}_{\mathcal{PA}}$ by a finite sorted signature $\mathcal{F}$. We usually omit stating the sorts in examples if they can be inferred from the context. 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 $\mathrm{root}(s)$. The root position of a term is denoted by ε. We write s^* for a tuple $s_1, \ldots, s_n$ of terms and extend notions from terms to tuples of terms component-wise. For an arbitrary set $\mathcal{E}$ of equations and terms s, t we write $s \to_{\mathcal{E}} t$ iff there exist an equation $u \approx v \in \mathcal{E}$, a substitution σ, and a position $p \in \mathcal{P}os(s)$ such that $s|_p = u\sigma$ and $t = s[t\sigma]_p$. The symmetric closure of $\to_{\mathcal{E}}$ is denoted by $\vdash\dashv_{\mathcal{E}}$, and the reflexive transitive closure of $\vdash\dashv_{\mathcal{E}}$ is denoted by $\sim_{\mathcal{E}}$. For two terms s, t we write $s \sim_{\mathcal{E}}^{<\varepsilon} t$ iff $s = f(s^*)$ and $t = f(t^*)$ such that $s^* \sim_{\mathcal{E}} t^*$.

The rewrite rules that we use have constraints on natural numbers that guard when a rewrite step may be performed.

Definition 2 (Syntax of $\mathcal{PA}$-constraints). *An* atomic $\mathcal{PA}$-constraint *has the form $s \simeq t$ or $s > t$ for terms $s, t \in \mathcal{T}(\mathcal{F}_{\mathcal{PA}}, \mathcal{V})$, or $k \mid s$ for some $k \in \mathbb{N} - \{0\}$ and $s \in \mathcal{T}(\mathcal{F}_{\mathcal{PA}}, \mathcal{V})$. The set of* $\mathcal{PA}$-constraints *is inductively defined as follows:*

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

The other Boolean connectives $\vee$, $\Rightarrow$, and $\Leftrightarrow$ are defined as usual. We will also use $\mathcal{PA}$-constraints of the form $s \geq t$, $s < t$, and $s \leq t$ as abbreviations for $s > t \vee s \simeq t$, $t > s$, and $t > s \vee t \simeq s$, respectively. We will write $s \not\simeq t$ for $\neg(s \simeq t)$, and similarly for the other predicates.

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

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

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

$\mathcal{PA}$-validity and $\mathcal{PA}$-satisfiability are decidable [15].

Now the rewrite rules that we consider are ordinary conditional rewrite rules together with a $\mathcal{PA}$-constraint C. The rewrite relation obtained by this kind of rules will be introduced in Definition 10.

Definition 4 (Conditional Constrained Rewrite Rules). *A* conditional constrained rewrite rule *has the form*

$$s_1 \to^* t_1, \ldots, s_n \to^* t_n \mid l \to r[\![C]\!]$$

where

1. *$l, r \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}_{\mathcal{PA}}, \mathcal{V})$ such that $\mathrm{root}(l) \in \mathcal{F}$,*
2. *$s_i, t_i \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}_{\mathcal{PA}}, \mathcal{V})$,*
3. *C is a $\mathcal{PA}$-constraint,*
4. *$\mathcal{V}(r) \subseteq \mathcal{V}(l) \cup \bigcup_{j=1}^{n} \mathcal{V}(t_j)$, and*
5. *$\mathcal{V}(s_i) \subseteq \mathcal{V}(l) \cup \bigcup_{j=1}^{i-1} \mathcal{V}(t_j)$ for all $1 \leq i \leq n$.*[1]

[1] Using the notation of [13], the last two conditions yield deterministic type 3 rules.

The difference between conditions and constraints in a rule is operational. Conditions need to be evaluated by recursively rewriting them, while constraints are checked using a decision procedure for $\mathcal{PA}$-validity. This distinction will be formalized in Definition 10. In a rule $l \to r[\![\top]\!]$ the constraint $\top$ will usually be omitted. For a set $\mathcal{R}$ of constrained rewrite rules, the set of *defined symbols* is given by $\mathcal{D}(\mathcal{R}) = \{f \mid f = \text{root}(l) \text{ for some } l \to r[\![C]\!] \in \mathcal{R}\}$. The set of *constructors* is $\mathcal{C}(\mathcal{R}) = \mathcal{F} - \mathcal{D}(\mathcal{R})$. Note that according to this definition, the symbols from $\mathcal{F}_{\mathcal{PA}}$ are considered to be neither defined symbols nor constructors. In the following we assume that $\mathcal{C}(\mathcal{R})$ does not contain any constructor with resulting sort `nat` (*signature condition*)[2].

Properties of non-free data structures will be modelled using *constructor equations* and *constructor rules*. Constructor equations need to be linear and regular.

Definition 5 (Constructor Equations, Identical Unique Variables). *Let $\mathcal{R}$ be a finite set of constrained rewrite rules. A* constructor equation *has the form $u \approx v$ for terms $u, v \in \mathcal{T}(\mathcal{C}(\mathcal{R}), \mathcal{V})$ such that $u \approx v$ has* identical unique variables *(is* i.u.v.*), i.e., u and v are linear and $\mathcal{V}(u) = \mathcal{V}(v)$.*

Similar to constrained rewrite rules, constrained constructor rules have a $\mathcal{PA}$-constraint that will guard when a rule is applicable.

Definition 6 (Constrained Constructor Rules). *Let $\mathcal{R}$ be a finite set of constrained rewrite rules. A* constrained constructor rule *has the form $l \to r[\![C]\!]$ for terms $l, r \in \mathcal{T}(\mathcal{C}(\mathcal{R}), \mathcal{V})$ and a $\mathcal{PA}$-constraint C such that* $\text{root}(l) \in \mathcal{C}(\mathcal{R})$ *and $\mathcal{V}(r) \subseteq \mathcal{V}(l)$.*

Again, constraints C of the form $\top$ will usually be omitted in constrained constructor rules. Constructor equations and constrained constructor rules give rise to the following rewrite relation. It is based on extended rewriting [14] but requires that the $\mathcal{PA}$-constraint of the constrained constructor rule is $\mathcal{PA}$-valid after being instantiated by the matcher. For this, we require that variables of sort `nat` are instantiated by terms over $\mathcal{F}_{\mathcal{PA}}$ in order to ensure that $\mathcal{PA}$-validity of the instantiated $\mathcal{PA}$-constraint can be decided by a decision procedure for $\mathcal{PA}$-validity.

Definition 7 ($\mathcal{PA}$-based Substitutions). *A substitution σ is* $\mathcal{PA}$-based *iff $\sigma(x) \in \mathcal{T}(\mathcal{F}_{\mathcal{PA}}, \mathcal{V})$ for all variables x of sort* `nat`.

Definition 8 (Constructor Rewrite Relation). *Let $\mathcal{E}$ be a finite set of constructor equations and let $\mathcal{S}$ be a finite set of constrained constructor rules. Then $s \to_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{S}} t$ iff there exist a constrained constructor rule $l \to r[\![C]\!] \in \mathcal{S}$, a position $p \in \mathcal{P}os(s)$, and a $\mathcal{PA}$-based substitution σ such that*

1. $s|_p \sim_{\mathcal{E}\cup\mathcal{PA}} l\sigma$,

[2] This restriction avoids “confusion” since it would otherwise be possible to introduce a constant `c` of sort `nat` and then add equations $\mathtt{c} \approx \mathtt{0}$ and $\mathtt{c} \approx \mathtt{1}$, implying $\mathtt{0} \approx \mathtt{1}$.

2. $C\sigma$ is $\mathcal{PA}$-valid, and
3. $t = s[r\sigma]_p$.

We write $s \to^{<\varepsilon}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{S}} t$ iff $s \to_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{S}} t$ at a position $p \neq \varepsilon$, and $s \overset{!}{\to}{}^{<\varepsilon}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{S}} t$ iff s reduces to t in zero or more $\to^{<\varepsilon}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{S}}$ steps such that t is a normal form w.r.t. $\to^{<\varepsilon}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{S}}$.

We combine constrained rewrite rules, constrained constructor rules, and constructor equations into a constrained equational system under certain conditions.

Definition 9 (Conditional Constrained Equational Systems (CCES)). *A* conditional constrained equational system (CCES) *has the form $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ for a finite set $\mathcal{R}$ of conditional constrained rewrite rules, a finite set $\mathcal{S}$ of constrained constructor rules, and a finite set $\mathcal{E}$ of constructor equations such that*

1. *$\mathcal{S}$ is right-linear, i.e., each variable occurs at most once in r for all $l \to r[\![C]\!] \in \mathcal{S}$,*
2. *$\sim_{\mathcal{E}\cup\mathcal{PA}}$ commutes over $\to_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{S}}$, i.e., the inclusion $\sim_{\mathcal{E}\cup\mathcal{PA}} \circ \to_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{S}} \subseteq \to_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{S}} \circ \sim_{\mathcal{E}\cup\mathcal{PA}}$ holds, and*
3. *$\to_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{S}}$ is convergent modulo $\sim_{\mathcal{E}\cup\mathcal{PA}}$, i.e., $\to_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{S}}$ is terminating and $\leftarrow^*_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{S}} \circ \to^*_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{S}} \subseteq \to^*_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{S}} \circ \sim_{\mathcal{E}\cup\mathcal{PA}} \circ \leftarrow^*_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{S}}$.*

Here, the commutation property intuitively states that if $s \sim_{\mathcal{E}\cup\mathcal{PA}} s'$ and $s' \to_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{S}} t'$, then $s \to_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{S}} t$ 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* [14, 7].

If $\mathcal{R}$ is unconditional (i.e., $n = 0$ for all conditional constrained rewrite rules $s_1 \to^* t_1, \ldots, s_n \to^* t_n \mid l \to r[\![C]\!]$ from $\mathcal{R}$) we will also speak of *constrained equational systems (CESs)*.

Some commonly used data structures and their specifications in our framework are listed in Figure 1. The rule marked by "$(*)$" is needed in order to make $\sim_{\mathcal{E}\cup\mathcal{PA}}$ commute over $\to_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{S}}$. The constructor $\langle\cdot\rangle$ used for sets and multisets creates a singleton set or multiset, respectively.

Finally, we can define the rewrite relation corresponding to a CES. The relation is an extension of the normalized rewrite relation used in [6], which in turn is based on [12]. The relation $\overset{\mathcal{S}}{\to}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}}$ needs to evaluate the conditions before a term may be reduced.

Definition 10 (Conditional Rewrite Relation). *Let $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ be a CCES. Then $\overset{\mathcal{S}}{\to}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}}$ is the least relation satisfying $s \overset{\mathcal{S}}{\to}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}} t$ iff there exist a conditional constraint rewrite rule $s_1 \to^* t_1, \ldots, s_n \to^* t_n \mid l \to r[\![C]\!]$ in $\mathcal{R}$, a position $p \in \mathcal{P}os(s)$, and a $\mathcal{PA}$-based substitution σ such that*

1. *$s|_p \overset{!}{\to}{}^{<\varepsilon}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{S}} \circ \sim^{<\varepsilon}_{\mathcal{E}\cup\mathcal{PA}} l\sigma$,*
2. *$C\sigma$ is $\mathcal{PA}$-valid,*
3. *$s_i\sigma \overset{\mathcal{S}}{\to}{}^*_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}} \circ \sim_{\mathcal{E}\cup\mathcal{PA}} t_i\sigma$ for all $1 \leq i \leq n$, and*
4. *$t = s[r\sigma]_p$.*

	Constructors	$\mathcal{E}$	$\mathcal{S}$
Lists	$\mathsf{nil}, \mathsf{cons}$		
Sorted lists	$\mathsf{nil}, \mathsf{cons}$		$\mathsf{cons}(x, \mathsf{cons}(y, zs))$ $\to \mathsf{cons}(y, \mathsf{cons}(x, zs))[\![x > y]\!]$
Multisets	$\emptyset, \mathsf{ins}$	$\mathsf{ins}(x, \mathsf{ins}(y, zs))$ $\approx \mathsf{ins}(y, \mathsf{ins}(x, zs))$	
Multisets	$\emptyset, \langle\cdot\rangle, \cup$	$x \cup (y \cup z) \approx (x \cup y) \cup z$ $x \cup y \approx y \cup x$	$x \cup \emptyset \to x$
Sets	$\emptyset, \mathsf{ins}$	$\mathsf{ins}(x, \mathsf{ins}(y, zs))$ $\approx \mathsf{ins}(y, \mathsf{ins}(x, zs))$	$\mathsf{ins}(x, \mathsf{ins}(y, zs))$ $\to \mathsf{ins}(x, zs)[\![x \simeq y]\!]$
Sets	$\emptyset, \langle\cdot\rangle, \cup$	$x \cup (y \cup z) \approx (x \cup y) \cup z$ $x \cup y \approx y \cup x$	$x \cup \emptyset \to x$ $x \cup x \to x$ $(x \cup x) \cup y \to x \cup y \quad (*)$
Sorted sets	$\emptyset, \mathsf{ins}$		$\mathsf{ins}(x, \mathsf{ins}(y, zs)$ $\to \mathsf{ins}(y, \mathsf{ins}(x, zs))[\![x > y]\!]$ $\mathsf{ins}(x, \mathsf{ins}(y, zs))$ $\to \mathsf{ins}(x, zs)[\![x \simeq y]\!]$

Fig. 1. Commonly used data structures.

The least relation satisfying these properties can be obtained by an inductive construction, similar to how this is done for ordinary conditional rewriting [13].

Example 11. This example continues Example 1. Assume we want to reduce the term $t = \mathsf{qsort}(\mathsf{ins}(1, \mathsf{ins}(3, \mathsf{ins}(1, \emptyset))))$ using $\xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}}$. Using the substitution

$$\sigma = \{x \mapsto 3, ys \mapsto \mathsf{ins}(1, \emptyset), yl \mapsto \mathsf{ins}(1, \emptyset), yh \mapsto \emptyset\}$$

we obtain $t \overset{!}{\to}{}^{<\varepsilon}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{S}} \mathsf{qsort}(\mathsf{ins}(1, \mathsf{ins}(3, \emptyset))) \sim^{<\varepsilon}_{\mathcal{E}\cup\mathcal{PA}} \mathsf{qsort}(\mathsf{ins}(x, ys))\sigma$ and thus $t \xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}} \mathsf{app}(\mathsf{qsort}(\mathsf{ins}(1, \emptyset)), \mathsf{cons}(3, \mathsf{qsort}(\emptyset)))$ using the third rule for qsort, *provided that* $\mathsf{split}(3, \mathsf{ins}(1, \emptyset)) \xrightarrow{\mathcal{S}}{}^{*}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}} \circ \sim_{\mathcal{E}\cup\mathcal{PA}} \langle \mathsf{ins}(1, \emptyset), \emptyset \rangle$. In order to verify this, we use the second split-rule. But in order to do so we first need to show that $\mathsf{split}(3, \emptyset) \xrightarrow{\mathcal{S}}{}^{*}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}} \circ \sim_{\mathcal{E}\cup\mathcal{PA}} \langle \emptyset, \emptyset \rangle$, which is easily obtained by the first rule for split. Continuing the reduction of $\mathsf{app}(\mathsf{qsort}(\mathsf{ins}(1, \emptyset)), \mathsf{cons}(3, \mathsf{qsort}(\emptyset)))$ eventually produces the result $\mathsf{cons}(1, \mathsf{cons}(3, \mathsf{nil}))$. $\Diamond$

We now show that whenever $s \sim_{\mathcal{E}\cup\mathcal{PA}} s'$ and $s \xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}} t$, then $s' \xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}} t'$ for some $t' \sim_{\mathcal{E}\cup\mathcal{PA}} t$, i.e., we show that $\sim_{\mathcal{E}\cup\mathcal{PA}}$ commutes over $\xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}}$.

Lemma 12. *Let $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ be a CCES. Then $\sim_{\mathcal{E}\cup\mathcal{PA}} \circ \xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}} \subseteq \xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}} \circ \sim_{\mathcal{E}\cup\mathcal{PA}}$. Furthermore, the $\xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}}$ steps are performed using the same conditional constrained rewrite rule and $\mathcal{PA}$-based substitution.*

Proof. We show that $s \xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}} t$ and $s' \sim_{\mathcal{E}\cup\mathcal{PA}} s$ implies $s' \xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}} t'$ for some $t' \sim_{\mathcal{E}\cup\mathcal{PA}} t$. Thus, assume $s \xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}} t$. This means that $s = C[f(u^*)]$ for some context C with $f(u^*) \overset{!}{\rightarrow}^{<\varepsilon}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{S}} \circ \sim^{<\varepsilon}_{\mathcal{E}\cup\mathcal{PA}} l\sigma$ for some constrained rewrite rule $l \to r[\![D]\!] \in \mathcal{R}$ and some $\mathcal{PA}$-based substitution σ such that $D\sigma$ is $\mathcal{PA}$-valid and $t = C[r\sigma]$. Since $s \sim_{\mathcal{E}\cup\mathcal{PA}} s'$ and all equations in $\mathcal{E} \cup \mathcal{PA}$ are i.u.v. and do not contain symbols from $\mathcal{D}(\mathcal{R})$, an application of [6, Lemma 5] implies $s' = C'[f(u'^*)]$ for some context C' with $C' \sim_{\mathcal{E}\cup\mathcal{PA}} C$ and $u'^* \sim_{\mathcal{E}\cup\mathcal{PA}} u^*$. Therefore, $f(u'^*) \overset{!}{\rightarrow}^{<\varepsilon}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{S}} \circ \sim^{<\varepsilon}_{\mathcal{E}\cup\mathcal{PA}} l\sigma$ by [5, Lemma 12.2] and we can use the substitution σ to rewrite $s' = C'[f(u'^*)]$ to $t' = C'[r\sigma] \sim_{\mathcal{E}\cup\mathcal{PA}} C[r\sigma] = t$. $\square$

3 Termination and Operational Termination

Termination of a CCES means that there is no term that starts an infinite $\xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}}$ reduction, i.e., that the relation $\xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}}$ is well-founded. As is well-known, termination is not the only crucial property of conditional rewriting. In order to get a decidable rewrite relation it additionally has to be ensured that evaluation of the conditions does not loop. As argued in [11], the notion of *operational termination* is a natural choice for this property since it better captures the behavior of actual implementations of rewriting than other commonly used notions like *effective termination* [13].

As in [11], the recursive nature of conditional rewriting is reflected in an inference system which aims at proving that $s \xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}} t$ or $s \xrightarrow{\mathcal{S}}{}^*_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}} t$. Then operational termination is characterized by the absence of infinite proof trees for this inference system.

Definition 13 (Proof Trees). *Let $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ be a CCES. The set of (finite)* proof trees *for $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ and the* head *of a proof tree are inductively defined as follows.*

1. *An* open goal *G, where G is either $s \to t$ or $s \to^* t$ for some terms s, t, is a proof tree. In this case* head$(G) = G$ *is the head of the proof tree.*
2. *A* derivation tree*, denoted by*

$$T = \frac{T_1 \quad \cdots \quad T_n}{G}(\Delta)$$

is a proof tree, where G is as in the first case, Δ is one of the derivation rules in Figure 2, and $T_1, \ldots, T_n$ are proof trees such that

$$\frac{\text{head}(T_1) \quad \cdots \quad \text{head}(T_n)}{G}$$

is an instance of Δ. In this case, head$(T) = G$.

A proof tree is closed *iff it does not contain any open goals.*

Example 14. We again consider the CCES for quicksort from Examples 1 and 11. Then $\mathsf{qsort}(\mathsf{ins}(1, \mathsf{ins}(3, \mathsf{ins}(1, \emptyset)))) \to \mathsf{app}(\mathsf{qsort}(\mathsf{ins}(1, \emptyset)), \mathsf{cons}(3, \mathsf{qsort}(\emptyset)))$ is an open goal and

$$(\mathsf{Refl}) \quad \dfrac{}{s \to^* t}$$

$$\text{if } s \sim_{\mathcal{E}\cup\mathcal{PA}} t$$

$$(\mathsf{Tran}) \quad \dfrac{s \to t \qquad t \to^* u}{s \to^* u}$$

$$(\mathsf{Repl}) \quad \dfrac{s_1\sigma \to^* t_1\sigma \qquad \cdots \qquad s_n\sigma \to^* t_n\sigma}{s \to t}$$

$$\begin{array}{l} \text{if } s_1 \to^* t_1, \ldots, s_n \to^* t_n \mid l \to r[\![C]\!] \in \mathcal{R}, \\ \quad p \in \mathcal{P}os(s), \\ \quad \sigma \text{ is } \mathcal{PA}\text{-based}, \\ \quad s|_p \overset{!}{\to}{}^{<\varepsilon}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{S}} \circ \sim^{<\varepsilon}_{\mathcal{E}\cup\mathcal{PA}} l\sigma, \\ \quad C\sigma \text{ is } \mathcal{PA}\text{-valid, and} \\ \quad t = s[r\sigma]_p. \end{array}$$

Fig. 2. Derivation rules.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathsf{split}(3,\emptyset) \to \langle\emptyset,\emptyset\rangle}\,(\mathsf{Repl}) \quad \dfrac{}{\langle\emptyset,\emptyset\rangle \to^* \langle\emptyset,\emptyset\rangle}\,(\mathsf{Refl})}{\mathsf{split}(3,\emptyset) \to^* \langle\emptyset,\emptyset\rangle}\,(\mathsf{Tran})}{\mathsf{split}(3,\mathsf{ins}(1,\emptyset)) \to \langle\mathsf{ins}(1,\emptyset),\emptyset\rangle}\,(\mathsf{Repl}) \quad \dfrac{}{\langle\mathsf{ins}(1,\emptyset),\emptyset\rangle \to^* \langle\mathsf{ins}(1,\emptyset),\emptyset\rangle}\,(\mathsf{Refl})}{\mathsf{split}(3,\mathsf{ins}(1,\emptyset)) \to^* \langle\mathsf{ins}(1,\emptyset),\emptyset\rangle}\,(\mathsf{Tran})}{\mathsf{qsort}(\mathsf{ins}(1,\mathsf{ins}(3,\mathsf{ins}(1,\emptyset)))) \to \mathsf{app}(\mathsf{qsort}(\mathsf{ins}(1,\emptyset)),\mathsf{cons}(3,\mathsf{qsort}(\emptyset)))}\,(\mathsf{Repl})$$

is a closed proof tree with this goal as its head. ◊

Now an infinite proof tree is defined to be a sequence of proof trees such that each member of this sequence can be obtained from its immediate predecessor by expanding one or more open goals.

Definition 15 (Prefixes of Proof Trees, Infinite Proof Trees). *A proof tree T is a* prefix *of a proof tree T', written $T \subset T'$, if there are one or more open goals $G_1, \ldots, G_n$ in T such that T' is obtained from T by replacing each G_i by a derivation tree T_i with* $\text{head}(T_i) = G_i$*. An* infinite proof tree *is an infinite sequence $\{T_i\}_{i \geq 0}$ of finite proof trees such that $T_i \subset T_{i+1}$ for all $i \geq 0$.*

The following notion of *well-formed proof trees* captures the operational behavior of a rewrite engine that evaluates the conditions of a rewrite rule from left to right.

Definition 16 (Well-formed Proof Trees). *A proof tree T is* well-formed *if it is either an open goal, a closed proof tree, or a derivation tree of the form*

$$\frac{T_1 \qquad \cdots \qquad T_n}{G}(\Delta)$$

where T_j is a well-formed proof tree for all $1 \leq j \leq n$ and there is an $i \leq n$ such that T_i is not closed, T_j is closed for all $j < i$, and T_k is an open goal for all $k > i$. An infinite proof tree is well-formed *if it consists of well-formed proof trees.*

As mentioned above, *operational termination* is characterized by the absence of infinite well-formed proof trees.

Definition 17 (Operational Termination). *The CCES $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ is* operationally terminating *iff there are no infinite well-formed proof trees.*

For CESs, the notions of termination and operational termination coincide.

Lemma 18. *Let $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ be a CES. Then $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ is operationally terminating iff $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ is terminating.*

Proof. Let $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ be a CES.

"⇒": Assume $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ is not terminating and let

$$s_0 \overset{\mathcal{S}}{\to}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}} s_1 \overset{\mathcal{S}}{\to}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}} s_2 \overset{\mathcal{S}}{\to}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}} \cdots$$

be an infinite rewrite sequence. We construct an infinite proof tree $\{T_i\}_{i \geq 0}$ as follows:

$$T_0 = s_0 \to^* t \quad \text{for some arbitrary term } t \qquad \text{(an open goal)}$$

$$T_{i+1} = \frac{\dfrac{}{s_i \to s_{i+1}}(\mathsf{Repl}) \qquad s_{i+1} \to^* t}{T_i}(\mathsf{Tran})$$

Here, T_i is extended at its (only) open goal $s_i \to^* t$. Therefore, $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ is not operationally terminating.

"⇐": Assume $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ is not operationally terminating. Thus, there exists an infinite proof tree $\{T_i\}_{i \geq 0}$. Since $\mathcal{R}$ is unconditional, the shared head of the T_i has the form $s_0 \to^* t$ for some terms s_0, t. Furthermore, the rule (Tran) is applied to this head. The left subgoal thus generated is closed using (Repl) before the right subgoal can be expanded further.

$$\frac{\dfrac{}{s_0 \to s_1}(\mathsf{Repl}) \qquad s_1 \to^* t}{s_0 \to^* t}(\mathsf{Tran})$$

Thus, we obtain $s_0 \xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}} s_1$. Applying the same argument to the subgoal $s_1 \to^* t$ and continuing in the same fashion afterwards, we obtain the infinite rewrite sequence

$$s_0 \xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}} s_1 \xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}} s_2 \xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{R}} \cdots$$

Thus, $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ is not terminating. □

4 Elimination of Conditions

In order to show operational termination of a CCES $(\mathcal{R}, \mathcal{S}, \mathcal{E})$, we transform it into a CES $(\mathcal{U}(\mathcal{R}), \mathcal{S}, \mathcal{E})$ such that $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ and $(\mathcal{U}(\mathcal{R}), \mathcal{S}, \mathcal{E})$ are equivalent w.r.t. operational termination. We then check for termination of $(\mathcal{U}(\mathcal{R}), \mathcal{S}, \mathcal{E})$, which, by Lemma 18, is equivalent to operational termination of $(\mathcal{U}(\mathcal{R}), \mathcal{S}, \mathcal{E})$. Our transformation generalizes the classical one for ordinary conditional rewriting (see, e.g., [13, Definition 7.2.48]) to rewriting with equations, normalization, and constraints. An extension of the classical transformation to context-sensitive rewriting with equations was proposed in [3]. Our presentation is influenced by that paper.

Definition 19 (Transformation $\mathcal{U}$). *Let $\rho : s_1 \to^* t_2, \ldots, s_n \to^* t_n \mid l \to r[\![C]\!]$ be a conditional constrained rewrite rule. Then $\mathcal{U}(\rho)$ is defined by*

$$\begin{array}{lll}
\textit{if } n = 0 \textit{ then } \mathcal{U}(\rho) = \{\ \rho\ \} & & \\
\textit{if } n > 0 \textit{ then } \mathcal{U}(\rho) = \{\ l \to U_1^\rho(s_1, x_1^*)[\![C]\!]\ \} \cup & & (1)\\
\qquad\qquad\qquad\qquad \{\ U_{i-1}^\rho(t_{i-1}, x_{i-1}^*) \to U_i^\rho(s_i, x_i^*)[\![C]\!] \mid 2 \le i \le n\ \} \cup & & (2)\\
\qquad\qquad\qquad\qquad \{\ U_n^\rho(t_n, x_n^*) \to r[\![C]\!]\ \} & & (3)
\end{array}$$

Here, the U_i^ρ are fresh function symbols and, for $1 \le i \le n$, the expression x_i^* denotes the sorted list of variables in the set $\mathcal{V}(l) \cup \mathcal{V}(t_1) \cup \ldots \cup \mathcal{V}(t_{i-1})$ according to some fixed order on the set $\mathcal{V}$ of all variables. For a finite set $\mathcal{R}$ of conditional constrained rewrite rules we let $\mathcal{U}(\mathcal{R}) = \bigcup_{\rho \in \mathcal{R}} \mathcal{U}(\rho)$.

Example 20. Continuing Examples 1, 11, and 14 we get the following unconditional constrained rewrite rules.

$$\begin{array}{rll}
\mathsf{app}(\mathsf{nil}, zs) & \to zs & \\
\mathsf{app}(\mathsf{cons}(x, ys), zs) & \to \mathsf{cons}(x, \mathsf{app}(ys, zs)) & \\
\mathsf{split}(x, \emptyset) & \to \langle \emptyset, \emptyset \rangle & \\
\mathsf{split}(x, \mathsf{ins}(y, zs)) & \to \mathsf{U}_1(\mathsf{split}(x, zs), x, y, zs) & [\![x > y]\!]\\
\mathsf{U}_1(\langle zl, zh \rangle, x, y, zs) & \to \langle \mathsf{ins}(y, zl), zh \rangle & [\![x > y]\!]\\
\mathsf{split}(x, \mathsf{ins}(y, zs)) & \to \mathsf{U}_2(\mathsf{split}(x, zs), x, y, zs) & [\![x \not> y]\!]\\
\mathsf{U}_2(\langle zl, zh \rangle, x, y, zs) & \to \langle zl, \mathsf{ins}(y, zh) \rangle & [\![x \not> y]\!]\\
\mathsf{qsort}(\emptyset) & \to \mathsf{nil} & \\
\mathsf{qsort}(\mathsf{ins}(x, ys)) & \to \mathsf{U}_3(\mathsf{split}(x, ys), x, ys) & \\
\mathsf{U}_3(\langle yl, yh \rangle, x, ys) & \to \mathsf{app}(\mathsf{qsort}(yl), \mathsf{cons}(x, \mathsf{qsort}(yh))) &
\end{array}$$

Here, we did not label the function symbols U_i with the rule they were created for in order to ease readability. A termination proof for this system in contained in [5, Appendix D.3]. $\Diamond$

In order to show that $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ is operationally terminating if $(\mathcal{U}(\mathcal{R}), \mathcal{S}, \mathcal{E})$ is operationally terminating, we need the following lemma.

Lemma 21. *For any well-formed proof tree T for $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ whose head goal is either $s \to t$ or $s \to^* t$, there exists a well-formed proof tree $\beta(T)$ for $(\mathcal{U}(\mathcal{R}), \mathcal{S}, \mathcal{E})$ whose head goal is $s \to^* t$. Furthermore, if $T \subset T'$ for some T', then $\beta(T) \subset \beta(T')$.*

Before proving Lemma 21 we make two preliminary remarks about proof trees. These properties will be used freely in the proof of Lemma 21.

Property 22. If we are given the proof tree

$$\frac{T_1 \qquad \cdots \qquad T_n}{s \to t}$$

and a term $s' \sim_{\mathcal{E} \cup \mathcal{PA}} s$, then we can construct the proof tree

$$\frac{T_1 \qquad \cdots \qquad T_n}{s' \to t'}$$

where $t' \sim_{\mathcal{E} \cup \mathcal{PA}} t$ is given by Lemma 12. $\square$

Property 23. Assume we are given the proof tree

$$\dfrac{\dfrac{T_1}{s_0 \to s_1} \qquad \dfrac{\dfrac{T_2}{s_1 \to s_2} \qquad \begin{array}{c}\dfrac{\dfrac{T_n}{s_{n-1} \to s_n} \qquad \dfrac{}{s_n \to^* t}\,\textsf{(Refl)}}{}\,\textsf{(Tran)} \\ \vdots \end{array}}{s_1 \to^* t}\,\textsf{(Tran)}}{s \to^* t}\,\textsf{(Tran)}$$

where $s_0 = s$ and $s_n \sim_{\mathcal{E} \cup \mathcal{PA}} t$. Given a term $s' \sim_{\mathcal{E} \cup \mathcal{PA}} s$, we can construct the proof tree

$$\dfrac{\dfrac{T_1}{\widetilde{s_0} \to \widetilde{s_1}} \qquad \dfrac{\dfrac{T_2}{\widetilde{s_1} \to \widetilde{s_2}} \qquad \begin{array}{c}\dfrac{\dfrac{T_n}{\widetilde{s_{n-1}} \to \widetilde{s_n}} \qquad \dfrac{}{\widetilde{s_n} \to^* t}\,\textsf{(Refl)}}{}\,\textsf{(Tran)} \\ \vdots \end{array}}{\widetilde{s_1} \to^* t}\,\textsf{(Tran)}}{s' \to^* t}\,\textsf{(Tran)}$$

where $\widetilde{s_0} = s'$ and, for any term u, the expression $\widetilde{u}$ denotes some term with $\widetilde{u} \sim_{\mathcal{E} \cup \mathcal{PA}} u$. Here, the terms $\widetilde{s_i}$ are given by Lemma 12. Notice that $\widetilde{s_n} \sim_{\mathcal{E} \cup \mathcal{PA}} t$ since $\widetilde{s_n} \sim_{\mathcal{E} \cup \mathcal{PA}} s_n$ and $s_n \sim_{\mathcal{E} \cup \mathcal{PA}} t$. $\square$

Proof of Lemma 21. For the proof of the lemma, the construction of $\beta(T)$ is done by induction on the structure of T. There are two cases, depending on whether the head goal of T is of the form $s \to^* t$ or $s \to t$.

I. The head goal is $s \to^* t$:
First, we assume that T is closed. Then, T has the shape

$$\dfrac{\dfrac{T_1}{s_0 \to s_1} \qquad \dfrac{\dfrac{T_2}{s_1 \to s_2} \qquad \begin{array}{c} \dfrac{\dfrac{T_n}{s_{n-1} \to s_n} \qquad \dfrac{}{s_n \to^* t}\,\textsf{(Refl)}}{}\,\textsf{(Tran)} \\ \vdots \end{array}}{s_1 \to^* t}\,\textsf{(Tran)}}{s \to^* t}\,\textsf{(Tran)}$$

where $s_0 = s$ and $s_n \sim_{\mathcal{E}\cup\mathcal{PA}} t$. By the induction hypotheses, we can assume that each subtree

$$U_i = \dfrac{T_i}{s_{i-1} \to s_i}$$

has a transformed tree $\beta(U_i)$ of the form

$$\dfrac{\dfrac{T_i^1}{s_{i-1} \to s_i^1} \qquad \dfrac{\dfrac{T_i^2}{s_i^1 \to s_i^2} \qquad \begin{array}{c} \dfrac{\dfrac{T_i^{k_i}}{s_i^{k_i-1} \to s_i^{k_i}} \qquad \dfrac{}{s_i^{k_i} \to^* s_i}\,\textsf{(Refl)}}{}\,\textsf{(Tran)} \\ \vdots \end{array}}{s_i^1 \to^* s_i}\,\textsf{(Tran)}}{s_{i-1} \to^* s_i}\,\textsf{(Tran)}$$

Now, the proof tree $\beta(T)$ is built as follows.

$$\dfrac{\dfrac{T_1^1}{\widetilde{s_0} \to \widetilde{s_1^1}} \qquad \dfrac{\dfrac{T_1^2}{\widetilde{s_1^1} \to \widetilde{s_1^2}} \qquad \begin{array}{c} \dfrac{\dfrac{T_1^{k_1}}{\widetilde{s_1^{k_1-1}} \to \widetilde{s_1^{k_1}}} \qquad \dfrac{\dfrac{T_2^1}{\widetilde{s_1^{k_1}} \to \widetilde{s_2^1}} \qquad \begin{array}{c} \dfrac{\dfrac{T_n^1}{\widetilde{s_{n-1}^{k_n}} \to \widetilde{s_n^1}} \qquad \dfrac{\dfrac{T_n^2}{\widetilde{s_n^1} \to \widetilde{s_n^2}} \qquad \begin{array}{c} \dfrac{\dfrac{T_n^{k_n}}{\widetilde{s_n^{k_n-1}} \to \widetilde{s_n^{k_n}}} \qquad \dfrac{}{\widetilde{s_n^{k_n}} \to^* t}\,\textsf{(Refl)}}{\widetilde{s_n^{k_n-1}} \to^* t}\,\textsf{(Tran)} \\ \vdots \\ \vdots \end{array}}{\widetilde{s_n^1} \to^* t}\,\textsf{(Tran)}}{\widetilde{s_{n-1}^{k_n}} \to^* t}\,\textsf{(Tran)} \\ \vdots \\ \vdots \end{array}}{\widetilde{s_1^{k_1}} \to^* t}\,\textsf{(Tran)}}{\widetilde{s_1^{k_1-1}} \to^* t}\,\textsf{(Tran)} \\ \vdots \end{array}}{\widetilde{s_1^1} \to^* t}\,\textsf{(Tran)}}{s \to^* t}\,\textsf{(Tran)}$$

If T is not closed since some leftmost T^i_j is not closed, then $\beta(T)$ needs to be cut at some level of T^i_j. In either case, $\beta(T)$ is a well-formed proof tree if T is well-formed and $\beta(T) \subset \beta(T')$ if $T \subset T'$.

II. The head goal is $s \to t$:
Again, we first assume that T is closed. Then, it has the shape

$$\dfrac{\dfrac{S_1}{s_1\sigma \to^* t_1\sigma} \quad \cdots \quad \dfrac{S_n}{s_n\sigma \to^* t_n\sigma}}{s \to t}\ (\mathsf{Repl})$$

for some rule $\rho : s_1 \to^* t_1, \ldots, s_n \to^* t_n \mid l \to r[\![C]\!]$ from $\mathcal{R}$. In order to ease notation, we assume that the position in the (Repl) rule is $p = \varepsilon$, i.e., $s \xrightarrow{!}{}^{<\varepsilon}_{\mathcal{PA}\|\mathcal{E}\backslash\mathcal{S}} \circ \sim_{\mathcal{E}\cup\mathcal{PA}} l\sigma$ and $t = r\sigma$. If the constrained rewrite rule that is used is unconditional, then this rule is also present in $\mathcal{U}(\mathcal{R})$ and we obtain the following proof tree for $(\mathcal{U}(\mathcal{R}), \mathcal{S}, \mathcal{E})$:

$$\dfrac{\dfrac{}{s \to t}\ (\mathsf{Repl}) \qquad \dfrac{}{t \to^* t}\ (\mathsf{Refl})}{s \to^* t}\ (\mathsf{Tran})$$

Otherwise, $\mathcal{U}(\mathcal{R})$ contains rules of the form (1), (2), and (3) from Definition 19. From these, we construct proof trees for $(\mathcal{U}(\mathcal{R}), \mathcal{S}, \mathcal{E})$ with the following head goals:

$$\begin{array}{rl}
U^\rho_n(t_n, x^*_n)\sigma \to^* r\sigma & (G_n)\\
U^\rho_n(s_n, x^*_n)\sigma \to^* r\sigma & (H_n)\\
U^\rho_{n-1}(t_{n-1}, x^*_{n-1})\sigma \to^* r\sigma & (G_{n-1})\\
U^\rho_{n-1}(s_{n-1}, x^*_{n-1})\sigma \to^* r\sigma & (H_{n-1})\\
\vdots & \\
U^\rho_1(t_1, x^*_1)\sigma \to^* r\sigma & (G_1)\\
U^\rho_1(s_1, x^*_1)\sigma \to^* r\sigma & (H_1)\\
s \to^* t & (K)
\end{array}$$

For the following, note that $C\sigma$ is $\mathcal{PA}$-valid by assumption.

1. Proof tree for (G_n):
We can construct the proof tree

$$\dfrac{\dfrac{}{U^\rho_n(t_n, x^*_n)\sigma \to r\sigma}\ (\mathsf{Repl}) \qquad \dfrac{}{r\sigma \to^* r\sigma}\ (\mathsf{Refl})}{U^\rho_n(t_n, x^*_n)\sigma \to^* r\sigma}\ (\mathsf{Tran})$$

by using rule (3) from Definition 19.
2. Proof tree for (H_k) using the proof tree for (G_k):
We assume that we have already constructed a proof tree T_k for the goal $(G_k) = U^\rho_k(t_k\sigma, x^*_k\sigma) \to^* r\sigma$. By induction on the tree structure, we can furthermore assume that the subtree

$$U_k = \dfrac{S_k}{s_k\sigma \to^* t_k\sigma}$$

has a transformed tree $\beta(U_k)$ of the form

$$\dfrac{\dfrac{T_k^1}{u_0 \to u_1} \qquad \dfrac{\dfrac{T_k^2}{u_1 \to u_2} \quad \begin{array}{c} \dfrac{\dfrac{T_k^l}{u_{l-1} \to u_l} \quad \dfrac{}{u_l \to^* t_k\sigma}\,(\mathsf{Refl})}{}\,(\mathsf{Tran}) \\ \vdots \end{array}}{u_1 \to^* t_k\sigma}\,(\mathsf{Tran})}{s_k\sigma \to^* t_k\sigma}\,(\mathsf{Tran})$$

where $u_0 = s_k\sigma$ and $u_l \sim_{\mathcal{E}\cup\mathcal{PA}} t_k\sigma$. Then, we can construct a proof tree for the goal $U_k^\rho(s_k\sigma, x_k^*\sigma) \to^* r\sigma$ as follows:

$$\dfrac{\dfrac{T_k^1}{u'_0 \to u'_1} \qquad \dfrac{\dfrac{T_k^2}{u'_1 \to u'_2} \quad \begin{array}{c} \dfrac{\dfrac{T_k^l}{u'_{l-1} \to u'_l} \quad \dfrac{T_k}{U_k^\rho(u_l, x_k^*\sigma) \to^* r\sigma}}{u'_{l-1} \to^* r\sigma}\,(\mathsf{Tran}) \\ \vdots \end{array}}{u'_1 \to^* r\sigma}\,(\mathsf{Tran})}{U_k^\rho(s_k\sigma, x_k^*\sigma) \to^* r\sigma}\,(\mathsf{Tran})$$

where $u'_i = U_k^\rho(u_i, x_k^*\sigma)$.

3. Proof tree for (G_{k-1}) using the proof tree for (H_k):
We assume that we have already constructed a proof tree T_k for the goal $(H_k) = U_k^\rho(s_k\sigma, x_k^*\sigma) \to^* r\sigma$. Then, we can construct a proof tree for the goal $U_{k-1}^\rho(t_{k-1}\sigma, x_{k-1}^*\sigma) \to^* r\sigma$ as follows:

$$\dfrac{\dfrac{}{U_{k-1}^\rho(t_{k-1}\sigma, x_{k-1}^*\sigma) \to U_k^\rho(s_k\sigma, x_k^*\sigma)}\,(\mathsf{Repl}) \qquad T_k}{U_{k-1}^\rho(t_{k-1}\sigma, x_{k-1}^*\sigma) \to^* r\sigma}\,(\mathsf{Tran})$$

where the (Repl) step uses rule (2) from Definition 19.

4. Proof tree for (K) using the proof tree for (H_1):
We assume that we have already constructed a proof tree T_1 for the goal $(H_1) = U_1^\rho(s_1\sigma, x_1^*\sigma) \to^* r\sigma$. Then, we can construct a proof tree for the goal $s \to^* t$ as follows:

$$\dfrac{\dfrac{}{s \to U_1^\rho(s_1\sigma, x_1^*\sigma)}\,(\mathsf{Repl}) \qquad T_1}{s \to^* t}\,(\mathsf{Tran})$$

where the (Repl) step uses rule (1) from Definition 19.

As in case I., if the original proof tree is not closed, then the transformed tree is cut at some level. In either case, $\beta(T)$ is well-formed if T is well-formed and $\beta(T) \subset \beta(T')$ if $T \subset T'$. □

Theorem 24. *If $(\mathcal{U}(\mathcal{R}), \mathcal{S}, \mathcal{E})$ is operationally terminating, then $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ is operationally terminating.*

Proof. Assume $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ is not operationally terminating. Thus, there exists an infinite proof tree $\{T_i\}_{i \geq 0}$ for $(\mathcal{R}, \mathcal{S}, \mathcal{E})$. By Lemma 21, we obtain an infinite sequence $\{\beta(T_i)\}_{i \geq 0}$ of proof trees for $(\mathcal{U}(\mathcal{R}), \mathcal{S}, \mathcal{E})$. Additionally, $T_i \subset T_{i+1}$ implies $\beta(T_i) \subset \beta(T_{i+1})$ for all $i \geq 0$. Therefore, $\{\beta(T_i)\}_{i \geq 0}$ is an infinite proof tree for $(\mathcal{U}(\mathcal{R}), \mathcal{S}, \mathcal{E})$, which is thus not operationally terminating. □

Finally, we obtain the desired result that operational termination of $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ is implied by termination of $(\mathcal{U}(\mathcal{R}), \mathcal{S}, \mathcal{E})$.

Corollary 25. *If $(\mathcal{U}(\mathcal{R}), \mathcal{S}, \mathcal{E})$ is terminating, then $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ is operationally terminating.*

Proof. By Lemma 18, termination of $(\mathcal{U}(\mathcal{R}), \mathcal{S}, \mathcal{E})$ implies operational termination of $(\mathcal{U}(\mathcal{R}), \mathcal{S}, \mathcal{E})$ since $\mathcal{U}(\mathcal{R})$ is unconditional. Thus, $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ is operationally terminating by Theorem 24. □

Example 26. The following CCES specifies the sieve of Eratosthenes. $\mathsf{primes}(x)$ returns a list containing the prime numbers up to x. We have $\mathcal{S} = \mathcal{E} = \emptyset$.

$$\begin{array}{rll}
\mathsf{primes}(x) & \to \mathsf{sieve}(\mathsf{nats}(2, x)) & \\
\mathsf{nats}(x, y) & \to \mathsf{nil} & [\![x > y]\!] \\
\mathsf{nats}(x, y) & \to \mathsf{cons}(x, \mathsf{nats}(x + 1, y)) & [\![x \not> y]\!] \\
\mathsf{sieve}(\mathsf{nil}) & \to \mathsf{nil} & \\
\mathsf{sieve}(\mathsf{cons}(x, ys)) & \to \mathsf{cons}(x, \mathsf{sieve}(\mathsf{filter}(x, ys))) & \\
\mathsf{filter}(x, \mathsf{nil}) & \to \mathsf{nil} & \\
\mathsf{isdiv}(x, y) \to^* \mathsf{true} \mid \mathsf{filter}(x, \mathsf{cons}(y, zs)) & \to \mathsf{filter}(x, zs) & \\
\mathsf{isdiv}(x, y) \to^* \mathsf{false} \mid \mathsf{filter}(x, \mathsf{cons}(y, zs)) & \to \mathsf{cons}(y, \mathsf{filter}(x, zs)) & \\
\mathsf{isdiv}(x, 0) & \to \mathsf{true} & [\![x > 0]\!] \\
\mathsf{isdiv}(x, y) & \to \mathsf{false} & [\![x > y \wedge y > 0]\!] \\
\mathsf{isdiv}(x, x + y) & \to \mathsf{isdiv}(x, y) & [\![x > 0]\!]
\end{array}$$

Following Definition 19 we obtain $\mathcal{U}(\mathcal{R})$ as follows.

$$\begin{array}{rll}
\mathsf{primes}(x) & \to \mathsf{sieve}(\mathsf{nats}(2, x)) & \\
\mathsf{nats}(x, y) & \to \mathsf{nil} & [\![x > y]\!] \\
\mathsf{nats}(x, y) & \to \mathsf{cons}(x, \mathsf{nats}(x + 1, y)) & [\![x \not> y]\!] \\
\mathsf{sieve}(\mathsf{nil}) & \to \mathsf{nil} & \\
\mathsf{sieve}(\mathsf{cons}(x, ys)) & \to \mathsf{cons}(x, \mathsf{sieve}(\mathsf{filter}(x, ys))) & \\
\mathsf{filter}(x, \mathsf{nil}) & \to \mathsf{nil} & \\
\mathsf{filter}(x, \mathsf{cons}(y, zs)) & \to \mathsf{U}_1(\mathsf{isdiv}(x, y), x, y, zs) & \\
\mathsf{U}_1(\mathsf{true}, x, y, zs) & \to \mathsf{filter}(x, zs) & \\
\mathsf{filter}(x, \mathsf{cons}(y, zs)) & \to \mathsf{U}_2(\mathsf{isdiv}(x, y), x, y, zs) & \\
\mathsf{U}_2(\mathsf{false}, x, y, zs) & \to \mathsf{cons}(y, \mathsf{filter}(x, zs)) & \\
\mathsf{isdiv}(x, 0) & \to \mathsf{true} & [\![x > 0]\!] \\
\mathsf{isdiv}(x, y) & \to \mathsf{false} & [\![x > y \wedge y > 0]\!] \\
\mathsf{isdiv}(x, x + y) & \to \mathsf{isdiv}(x, y) & [\![x > 0]\!]
\end{array}$$

By Corollary 25 the CCES $(\mathcal{R}, \emptyset, \emptyset)$ is operationally terminating if the unconditional CCES $(\mathcal{U}(\mathcal{R}), \emptyset, \emptyset)$ is terminating. Termination of $(\mathcal{U}(\mathcal{R}), \emptyset, \emptyset)$ is shown in [5, Appendix F.2]. ◊

5 Conclusions and Future Work

We have proposed the notion of conditional constrained equational systems for modeling algorithms. Rewriting with these systems is based on normalized rewriting combined with evaluation of conditions and validity checking of instantiated constraints. Semantic data structures like finite sets, multisets, and sorted lists are modeled using constrained rewrite rules and equations on constructors. In this paper, constraints are Boolean combinations of atomic formulas from Presburger arithmetic. The conditional constrained equational systems discussed in this paper are a strict generalization of the equational systems presented in [6], which also use normalized rewriting but do not allow the use of conditions or constraints.

We show that the operational termination of such conditional constrained equational systems can be reduced to termination of unconditional constrained equational systems using a simple transformation. Powerful methods for showing termination of unconditional constrained equational rewrite systems are presented in the companion paper [5]. These methods can thus be used for showing operational termination of conditional constrained equational rewrite systems as well. Using this approach the operational termination of several nontrivial conditional constrained equational systems is shown in [5].

Operational termination is only one among several important properties of conditional constrained equational systems. We plan to study other properties as well, in particular confluence and sufficient completeness. The cover set method [16] for automatically generating an induction scheme from a function definition requires that the function definition is both (operationally) terminating and sufficiently complete. Developing checks for sufficient completeness along with the results from this paper and from [5] for showing operational termination would allow the development of automated methods for mechanizing proofs by induction for such function definitions. Results about decidability of inductive validity of conjectures as discussed in [8, 4] could also be extended. Orthogonal to this, we will investigate how the rewrite relation can be generalized by considering other built-in theories apart from Presburger arithmetic on natural numbers.

References

1. Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.
2. Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn Talcott. The Maude 2.0 system. In Robert Nieuwenhuis, editor, *Proceedings of the 14th Conference on Rewriting Techniques and Applications (RTA '03)*, volume 2706 of *Lecture Notes in Computer Science*, pages 76–87. Springer-Verlag, 2003.
3. Francisco Durán, Salvador Lucas, Claude Marché, José Meseguer, and Xavier Urbain. Proving operational termination of membership equational programs. *Higher-Order and Symbolic Computation*, 2007. To appear.

4. Stephan Falke and Deepak Kapur. Inductive decidability using implicit induction. In Miki Hermann and Andrei Voronkov, editors, *Proceedings of the 13th Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR '06)*, volume 4246 of *Lecture Notes in Artificial Intelligence*, pages 45–59. Springer-Verlag, 2006. An expanded version is Technical Report TR-CS-2006-04, available from `http://www.cs.unm.edu/research/tech-reports/`.
5. Stephan Falke and Deepak Kapur. Dependency pairs for rewriting with built-in numbers and semantic data structures. Technical Report TR-CS-2007-21, Department of Computer Science, University of New Mexico, 2007. Available from `http://www.cs.unm.edu/research/tech-reports/`.
6. Stephan Falke and Deepak Kapur. Dependency pairs for rewriting with non-free constructors. In Frank Pfenning, editor, *Proceedings of the 21st Conference on Automated Deduction (CADE '07)*, volume 4603 of *Lecture Notes in Artificial Intelligence*, pages 426–442. Springer-Verlag, 2007. An expanded version is Technical Report TR-CS-2007-07, available from `http://www.cs.unm.edu/research/tech-reports/`.
7. Jürgen Giesl and Deepak Kapur. Dependency pairs for equational rewriting. In Aart Middeldorp, editor, *Proceedings of the 12th Conference on Rewriting Techniques and Applications (RTA '01)*, volume 2051 of *Lecture Notes in Computer Science*, pages 93–108. Springer-Verlag, 2001. An expanded version is Technical Report TR-CS-2000-53, available from `http://www.cs.unm.edu/research/tech-reports/`.
8. Deepak Kapur, Jürgen Giesl, and Mahadevan Subramaniam. Induction and decision procedures. *Revista de la Real Academia de Ciencias, Serie A: Matemáticas*, 98(1):153–180, 2004.
9. Deepak Kapur and Hantao Zhang. An overview of rewrite rule laboratory (RRL). *Computers & Mathematics with Applications*, 29(2):91–114, 1995.
10. Hélène Kirchner and Pierre-Etienne Moreau. Promoting rewriting to a programming language: A compiler for nondeterministic rewrite programs in associative-commutative theories. *Journal of Functional Programming*, 11(2):207–251, 2001.
11. Salvador Lucas, Claude Marché, and José Meseguer. Operational termination of conditional term rewriting systems. *Information Processing Letters*, 95(4):446–453, 2005.
12. Claude Marché. Normalized rewriting: An alternative to rewriting modulo a set of equations. *Journal of Symbolic Computation*, 21(3):253–288, 1996.
13. Enno Ohlebusch. *Advanced Topics in Term Rewriting*. Springer-Verlag, 2002.
14. Gerald E. Peterson and Mark E. Stickel. Complete sets of reductions for some equational theories. *Journal of the ACM*, 28(2):233–264, 1981.
15. Mojzesz Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In *Comptes Rendus du Premier Congrès de Mathématiciens des Pays Slaves*, pages 92–101, 1929.
16. Hantao Zhang, Deepak Kapur, and Mukkai S. Krishnamoorthy. A mechanizable induction principle for equational specifications. In Ewing L. Lusk and Ross A. Overbeek, editors, *Proceedings of the 9th Conference on Automated Deduction (CADE '88)*, volume 310 of *Lecture Notes in Computer Science*, pages 162–181. Springer-Verlag, 1988.