Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 237 (2009) 75–90
www.elsevier.com/locate/entcs

Operational Termination of Conditional Rewriting with Built-in Numbers and Semantic Data Structures*

Stephan Falke[1] Deepak Kapur[2]

Department of Computer Science, University of New Mexico, Albuquerque, NM 87131, USA

Abstract

While ordinary conditional rewrite systems are more elegant than unconditional ones, they still have limited expressive power since semantic data structures, such as sets or multisets, cannot be modeled elegantly. Extending our work presented at RTA 2008 [9], the present paper defines a class of conditional rewrite systems that allows the use of semantic data structures and supports built-in natural numbers, including constraints taken from Presburger arithmetic. The framework is both expressive and natural. Rewriting is performed using a combination of normalized equational rewriting with recursive evaluation of conditions and validity checking of instantiated constraints.
Termination is one of the most important properties of any kind of rewriting. For conditional systems, 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 terminates. These properties are captured by the notion of *operational termination*. In this work, we show that operational termination for the class of conditional rewrite systems discussed above can be reduced to (regular) termination of unconditional systems using a syntactic transformation. Powerful methods for showing termination of unconditional systems are presented in [9].

Keywords: Conditional term rewriting, operational termination, semantic data structures

1 Introduction

Conditional term rewrite systems operating on free data structures provide a powerful framework for specifying algorithms. This approach has successfully been taken by the system Maude [4]. Many algorithms, however, 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

* Partially supported by NSF grant CCF-0541315.
[1] Email: spf@cs.unm.edu
[2] Email: kapur@cs.unm.edu

doi:10.1016/j.entcs.2009.03.036

associative, commutative, idempotent, and has the empty set as unit element. Such semantic data structures can be modeled using equational axioms.

Extending our work presented at CADE 2007 and RTA 2008 [7,9], the present paper introduces conditional constrained equational rewrite systems (CCESs) 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, and (iii) $\mathcal{E}$, a set of equations. Here, (ii) and (iii) are over the constructors of $\mathcal{R}$ and are used for modeling semantic data structures such that 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 Boolean combinations of atomic formulas of the form $s \simeq t$ and $s > t$ from Presburger arithmetic. Rewriting with such a system is performed using a combination of normalized rewriting [12] with evaluation of conditions and validity checking of instantiated constraints. Before matching a redex with the left side of a rule, the redex is first normalized with $\mathcal{S}$. Additionally, the rewrite step is only performed if the instantiated conditions of the rule can be established by recursively rewriting them and if the instantiated constraint of the rule is valid. The difference between conditions and constraints in a rule is thus operational.

Example 1.1 *This example shows a quicksort algorithm that takes a set and returns a list. It is a modification of an example from [2] that is widely used in the literature on conditional rewriting. Sets are constructed using* $\emptyset$ *and* ins*, where* ins *adds an element to a set. The semantics of sets is modeled using* $\mathcal{S}$ *and* $\mathcal{E}$ *as follows.*

$$\mathcal{E}: \quad \mathsf{ins}(x, \mathsf{ins}(y, zs)) \approx \mathsf{ins}(y, \mathsf{ins}(x, zs))$$

$$\mathcal{S}: \quad \mathsf{ins}(x, \mathsf{ins}(x, ys)) \to \mathsf{ins}(x, ys)$$

Quicksort is 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* $x \not> y$. ◇

One of the most important properties of a CCES is that a rewrite engine operating with it always terminates. For this, it has to be shown that the rewrite relation is well-founded and that the evaluation of the conditions terminates. These properties

can be characterized by the notion of *operational termination* [11].[3] The recursive nature of conditional rewriting is reflected in an inference systems for proving that a term s can be reduced to a term t, and operational termination is the property that this inference system does not allow infinite derivations.

The present paper shows that operational termination of a conditional system can be reduced to termination of an unconditional system using a syntactic transformation. This transformation is similar to the transformation used for ordinary conditional rewriting, see, e.g., [13, Definition 7.2.48]. Powerful methods based on dependency pairs for showing termination of unconditional systems are presented in [9], and in combination with the current paper these methods can be used for showing operational termination of CCESs as well.

This paper is organized as follows. In Section 2, the rewrite relation is defined. In Section 3, we formally define the notion of operational termination and show that termination and operational termination coincide for unconditional systems. Section 4 introduces a transformation from conditional systems into unconditional ones. We show that termination of the transformed system implies operational termination of the original system. The omitted proofs may be found in the full version of this paper [8], and [6] contains several nontrivial conditional systems whose operational termination can be shown by applying the transformation presented in this paper and using the termination techniques presented in [9].

2 Conditional Normalized 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 use an initial signature $\mathcal{F}_{\mathcal{PA}} = \{0, 1, +\}$ using only sort `nat`. Here, "$\mathcal{PA}$" stands for "Presburger Arithmetic". 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 $1 + \ldots + 1$ (with k occurrences of 1) by 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. The root symbol of a term s is denoted by $\mathrm{root}(s)$. The root position of a term is denoted by λ. For an arbitrary set $\mathcal{E}$ of equations and terms s, t we write $s \to_{\mathcal{E}} t$ 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 $t = s[v\sigma]_p$. 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 $s \sim_{\mathcal{E}}^{>\lambda} t$ iff $s = f(s_1, \ldots, s_n)$ and $t = f(t_1, \ldots, t_n)$ 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

[3] Another commonly used characterization is *effective termination*, see, e.g., [13]. However, as argued in [11], operational termination better captures the behaviour of actual rewrite engines.

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 [15].

Now the conditional rewrite rules considered are combined with $\mathcal{PA}$-constraints.

Definition 2.1 (Conditional Constrained Rewrite Rule) *A* conditional constrained rewrite rule *has the form* $s_1 \to^* t_1, \ldots, s_n \to^* t_n \mid l \to r[\![C]\!]$ *such that*

(i) $l, r \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}_{\mathcal{PA}}, \mathcal{V})$ *such that* $\mathrm{root}(l) \in \mathcal{F}$,

(ii) $s_i, t_i \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}_{\mathcal{PA}}, \mathcal{V})$,

(iii) C *is a* $\mathcal{PA}$*-constraint,*

(iv) $\mathcal{V}(r) \subseteq \mathcal{V}(l) \cup \bigcup_{j=1}^{n} \mathcal{V}(t_j)$, *and*

(v) $\mathcal{V}(s_i) \subseteq \mathcal{V}(l) \cup \bigcup_{j=1}^{i-1} \mathcal{V}(t_j)$ *for all* $1 \leq i \leq n$. [4]

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 2.7. In a rule $l \to r[\![\top]\!]$ the constraint $\top$ will be omitted. For a set $\mathcal{R}$ of constrained rewrite rules, the set of *defined symbols* is given by $\mathcal{D}(\mathcal{R}) = \{f \mid f = \mathrm{root}(l) \text{ for some } s_1 \to^* t_1, \ldots, s_n \to^* t_n \mid l \to r[\![C]\!] \in \mathcal{R}\}$. 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.

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

Definition 2.2 (Constructor Equations) *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 conditional constrained rewrite rules, constructor rules have a $\mathcal{PA}$-constraint that will guard when a rule is applicable.

Definition 2.3 (Constructor Rules) *A* constructor rule *is a rule* $l \to r[\![C]\!]$ *with* $l, r \in \mathcal{T}(\mathcal{C}(\mathcal{R}), \mathcal{V})$ *and a* $\mathcal{PA}$*-constraint* C *where* $\mathrm{root}(l) \in \mathcal{C}(\mathcal{R})$ *and* $\mathcal{V}(r) \subseteq \mathcal{V}(l)$.

Again, constraints C of the form $\top$ will be omitted in constructor rules. Constructor rules and equations give rise to the following rewrite relation. It is based on extended rewriting [14] but requires that the $\mathcal{PA}$-constraint of the 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. [5]

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

[5] This requirement can be relaxed slighlty by requiring that only those variables of sort `nat` that occur in the $\mathcal{PA}$-constraint need to be instantiated by terms over $\mathcal{F}_{\mathcal{PA}}$.

Definition 2.4 ($\mathcal{PA}$-based Substitutions) *Let σ be a substitution. Then 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 2.5 (Constructor Rewrite Relation) *Let $\mathcal{E}$ be a finite set of constructor equations and let $\mathcal{S}$ be a finite set of constructor rules. Then $s \to_{\mathcal{PA}\|\mathcal{E}\setminus\mathcal{S}} t$ iff there exist a 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*

(i) $s|_p \sim_{\mathcal{E}\cup\mathcal{PA}} l\sigma$, [6]

(ii) $C\sigma$ *is $\mathcal{PA}$-valid, and*

(iii) $t = s[r\sigma]_p$.

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

We combine conditional constrained rewrite rules and constructor rules and equations into a conditional constrained equational system (CCES).

Definition 2.6 (CCES) *A* 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 constructor rules, and a finite set $\mathcal{E}$ of constructor equations such that*

(i) $\sim_{\mathcal{E}\cup\mathcal{PA}}$ *commutes over $\to_{\mathcal{PA}\|\mathcal{E}\setminus\mathcal{S}}$, i.e., $\sim_{\mathcal{E}\cup\mathcal{PA}} \circ \to_{\mathcal{PA}\|\mathcal{E}\setminus\mathcal{S}} \subseteq \to_{\mathcal{PA}\|\mathcal{E}\setminus\mathcal{S}} \circ \sim_{\mathcal{E}\cup\mathcal{PA}}$, and*

(ii) $\to_{\mathcal{PA}\|\mathcal{E}\setminus\mathcal{S}}$ *is convergent modulo $\sim_{\mathcal{E}\cup\mathcal{PA}}$, i.e., $\to_{\mathcal{PA}\|\mathcal{E}\setminus\mathcal{S}}$ is terminating and we have $\leftarrow^*_{\mathcal{PA}\|\mathcal{E}\setminus\mathcal{S}} \circ \to^*_{\mathcal{PA}\|\mathcal{E}\setminus\mathcal{S}} \subseteq \to^*_{\mathcal{PA}\|\mathcal{E}\setminus\mathcal{S}} \circ \sim_{\mathcal{E}\cup\mathcal{PA}} \circ \leftarrow^*_{\mathcal{PA}\|\mathcal{E}\setminus\mathcal{S}}$.*

The commutation property intuitively states that if $s \sim_{\mathcal{E}\cup\mathcal{PA}} s'$ and $s' \to_{\mathcal{PA}\|\mathcal{E}\setminus\mathcal{S}} t'$, then $s \to_{\mathcal{PA}\|\mathcal{E}\setminus\mathcal{S}} t$ for some $t \sim_{\mathcal{E}\cup\mathcal{PA}} t'$. Thus, if $s \sim_{\mathcal{E}\cup\mathcal{PA}} s'$ and s is irreducible by $\to_{\mathcal{PA}\|\mathcal{E}\setminus\mathcal{S}}$, then s' is irreducible by $\to_{\mathcal{PA}\|\mathcal{E}\setminus\mathcal{S}}$ as well. If $\mathcal{S}$ does not already satisfy the commutation property then it might be achieved by adding *extended rules* [14,10]. It is in general hard to check the conditions on $\to_{\mathcal{PA}\|\mathcal{E}\setminus\mathcal{S}}$ automatically and an implementation might thus be restricted to some commonly used data structures for which these properties have been established beforehand. Several examples are listed in Figure 1. The rule "$(*)$" is needed in order to make $\sim_{\mathcal{E}\cup\mathcal{PA}}$ commute over $\to_{\mathcal{PA}\|\mathcal{E}\setminus\mathcal{S}}$. The constructor $\langle\cdot\rangle$ creates a singleton set or multiset, respectively.

If $\mathcal{R}$ is unconditional (i.e., $n = 0$ for all $s_1 \to^* t_1, \ldots, s_n \to^* t_n \mid l \to r[\![C]\!]$ in $\mathcal{R}$), a CCES will also be called a *CES* [9]. The rewrite relation corresponding to a CCES is an extension of the rewrite relation considered in [9].

Definition 2.7 (Conditional Rewrite Relation) *Let $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ be a CCES. The rewrite relation $\overset{\mathcal{S}}{\to}_{\mathcal{PA}\|\mathcal{E}\setminus\mathcal{R}}$ is the least relation satisfying $s \overset{\mathcal{S}}{\to}_{\mathcal{PA}\|\mathcal{E}\setminus\mathcal{R}} t$ 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*

(i) $s|_p \overset{!}{\to}^{>\lambda}_{\mathcal{PA}\|\mathcal{E}\setminus\mathcal{S}} \circ \sim^{>\lambda}_{\mathcal{E}\cup\mathcal{PA}} l\sigma$,

(ii) $C\sigma$ *is $\mathcal{PA}$-valid,*

[6] Recall that $\mathcal{PA}$ also denotes the set of equations introduced above.

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

Fig. 1. Commonly used data structures.

(iii) $s_i\sigma \xrightarrow{\mathcal{S}}{}^*_{\mathcal{PA}\|\mathcal{E}\setminus\mathcal{R}} \circ \sim_{\mathcal{E}\cup\mathcal{PA}} t_i\sigma$ *for all* $1 \le i \le n$, *and*

(iv) $t = s[r\sigma]_p$.

Notice that the restriction to $\mathcal{PA}$-based substitution enforces a kind of innermost rewriting for function symbols with resulting sort `nat`. The least relation satisfying Definition 2.7 can be obtained by an inductive construction, similarly to ordinary conditional rewriting (see, e.g., [13]).

Example 2.8 *Continuing Example 1.1 we now illustrate* $\xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\setminus\mathcal{R}}$. *Consider* $t = \mathsf{qsort}(\mathsf{ins}(1, \mathsf{ins}(3, \mathsf{ins}(1, \emptyset))))$ *and the* $\mathcal{PA}$*-based substitution* $\sigma = \{x \mapsto 3, ys \mapsto \mathsf{ins}(1, \emptyset), yl \mapsto \mathsf{ins}(1, \emptyset), yh \mapsto \emptyset\}$. *We have* $t \xrightarrow{!}{}^{>\lambda}_{\mathcal{PA}\|\mathcal{E}\setminus\mathcal{S}} \mathsf{qsort}(\mathsf{ins}(1, \mathsf{ins}(3, \emptyset))) \sim^{>\lambda}_{\mathcal{E}\cup\mathcal{PA}} \mathsf{qsort}(\mathsf{ins}(x, ys))\sigma$ *and thus* $t \xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\setminus\mathcal{R}} \mathsf{app}(\mathsf{qsort}(\mathsf{ins}(1, \emptyset)), \mathsf{cons}(3, \mathsf{qsort}(\emptyset)))$ *using the third rule for* qsort, *provided* $\mathsf{split}(3, \mathsf{ins}(1, \emptyset)) \xrightarrow{\mathcal{S}}{}^*_{\mathcal{PA}\|\mathcal{E}\setminus\mathcal{R}} \circ \sim_{\mathcal{E}\cup\mathcal{PA}} \langle \mathsf{ins}(1, \emptyset), \emptyset\rangle$. *In order to verify this, we use the second* split*-rule. For this, we first need to check that the instantiated constraint* $3 \not> 1$ *is* $\mathcal{PA}$*-valid. Furthermore we need to show that* $\mathsf{split}(3, \emptyset) \xrightarrow{\mathcal{S}}{}^*_{\mathcal{PA}\|\mathcal{E}\setminus\mathcal{R}} \circ \sim_{\mathcal{E}\cup\mathcal{PA}} \langle \emptyset, \emptyset\rangle$, *which is established by the first* split*-rule. Reducing* $\mathsf{app}(\mathsf{qsort}(\mathsf{ins}(1, \emptyset)), \mathsf{cons}(3, \mathsf{qsort}(\emptyset)))$ *eventually produces* $\mathsf{cons}(1, \mathsf{cons}(3, \mathsf{nil}))$. $\diamond$

It is shown in [8] that whenever $s \sim_{\mathcal{E}\cup\mathcal{PA}} s'$ and $s \xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\setminus\mathcal{R}} t$, then $s' \xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\setminus\mathcal{R}} t'$ for some $t' \sim_{\mathcal{E}\cup\mathcal{PA}} t$, i.e., $\sim_{\mathcal{E}\cup\mathcal{PA}}$ commutes over $\xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\setminus\mathcal{R}}$.

Lemma 2.9 *For any CCES $(\mathcal{R},\mathcal{S},\mathcal{E})$ we have $\sim_{\mathcal{E}\cup\mathcal{PA}} \circ \xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\setminus\mathcal{R}} \subseteq \xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\setminus\mathcal{R}} \circ \sim_{\mathcal{E}\cup\mathcal{PA}}$. Furthermore, the $\xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\setminus\mathcal{R}}$ steps can be performed using the same conditional constrained rewrite rule and $\mathcal{PA}$-based substitution.*

3 Termination and Operational Termination

Termination of a CCES means that there is no term that starts an infinite $\xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\setminus\mathcal{R}}$ reduction, i.e., that the relation $\xrightarrow{\mathcal{S}}_{\mathcal{PA}\|\mathcal{E}\setminus\mathcal{R}}$ 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 terminates. As argued in [11], the notion of *operational termination* is a natural choice for this since it better captures the behavior of actual rewrite engines than other commonly used notions like *effective termination* [13].

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

Definition 3.1 (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.*

(i) *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* $\mathrm{head}(G) = G$ *is the head of the proof tree.*

(ii) *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

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

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

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

Example 3.2 *We again consider the CCES for quicksort from Examples 1.1 and 2.8. 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*

$$\text{(Refl)}\quad \frac{}{s \to^* t} \quad \text{if } s \sim_{\mathcal{E}\cup\mathcal{PA}} t$$

$$\text{(Tran)}\quad \frac{s \to t \qquad t \to^* u}{s \to^* u}$$

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

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

Fig. 2. Derivation rules.

an open goal and

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

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

An infinite proof tree is a sequence of proof trees such that each member of the sequence is obtained from its immediate predecessor by expanding open goals.

Definition 3.3 (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* $\mathrm{head}(T_i) = G_i$. *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 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 3.4 (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

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

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

It can be shown that the notions of termination and operational termination coincide for unconditional systems [8].

Lemma 3.6 *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.*

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 operational termination of $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ is implied by operational termination of $(\mathcal{U}(\mathcal{R}), \mathcal{S}, \mathcal{E})$. We then check for termination of $(\mathcal{U}(\mathcal{R}), \mathcal{S}, \mathcal{E})$, which, by Lemma 3.6, is equivalent to operational termination of $(\mathcal{U}(\mathcal{R}), \mathcal{S}, \mathcal{E})$. The 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 [5]. Our presentation is influenced by that paper.

Definition 4.1 (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*

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

Here, the U_i^ρ are fresh function symbols and, for $1 \leq i \leq n$, 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 4.2 *Continuing Examples 1.1, 2.8, and 3.2 we get the following unconditional constrained rewrite rules.*

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

In order to ease readability we used simplified names for the function symbols U_i^ρ from Definition 4.1. Termination of this system is shown in [6, Appendix D.3]. ◇

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 make use of the following lemma.

Lemma 4.3 *There exists a mapping β from well-formed proof trees for $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ to well-formed proof trees for $(\mathcal{U}(\mathcal{R}), \mathcal{S}, \mathcal{E})$ such that for any well-formed proof tree T with head goal $s \to t$ or $s \to^* t$, $\beta(T)$ is well-formed and has $s \to^* t$ as head goal. Furthermore, if $T \subset T'$ for some T', then $\beta(T) \subset \beta(T')$.*

The following properties are used freely in the proof of Lemma 4.3.

Property 4.4 *Given the proof tree*

$$\dfrac{T_1 \quad \cdots \quad T_n}{s \to t}$$

and a term $s' \sim_{\mathcal{E} \cup \mathcal{PA}} s$, it is possible to construct the proof tree

$$\dfrac{T_1 \quad \cdots \quad T_n}{s' \to t'}$$

where $t' \sim_{\mathcal{E} \cup \mathcal{PA}} t$ is given by Lemma 2.9. □

Property 4.5 *Given the proof tree*

$$
\dfrac{\dfrac{T_1}{s_0 \to s_1} \quad \dfrac{\dfrac{T_2}{s_1 \to s_2} \quad \dfrac{\dfrac{T_n}{s_{n-1} \to s_n} \quad \dfrac{}{s_n \to^* t}\,(\mathsf{Refl})}{\vdots}\,(\mathsf{Tran})}{s_1 \to^* t}\,(\mathsf{Tran})}{s \to^* t}\,(\mathsf{Tran})
$$

with $s_0 = s$ and a term $s' \sim_{\mathcal{E}\cup\mathcal{PA}} s$, it is possible to construct the proof tree

$$\dfrac{\dfrac{T_1}{\widetilde{s_0} \to \widetilde{s_1}} \quad \dfrac{\dfrac{T_2}{\widetilde{s_1} \to \widetilde{s_2}} \quad \dfrac{\dfrac{T_n}{\widetilde{s_{n-1}} \to \widetilde{s_n}} \quad \dfrac{}{\widetilde{s_n} \to^* t}\,(\mathsf{Refl})}{\vdots}\,(\mathsf{Tran})}{\widetilde{s_1} \to^* t}\,(\mathsf{Tran})}{s' \to^* t}\,(\mathsf{Tran})$$

where $\widetilde{s_0} = s'$ and $\widetilde{s_i} \sim_{\mathcal{E}\cup\mathcal{PA}} s_i$ for all $0 \le i \le n$. Here, the $\widetilde{s_i}$ are given by Lemma 2.9. 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 $s_n \sim_{\mathcal{E}\cup\mathcal{PA}} t$. □

Proof of Lemma 4.3

Assume that T is a well-formed proof tree for $(\mathcal{R}, \mathcal{S}, \mathcal{E})$ whose head goal is either $s \to t$ or $s \to^* t$. 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 $s \to^* t$:
If the inference rule (Refl) is applied to $s \to^* t$ then we are immediately done. Otherwise, the inference rule (Tran) is applied to $s \to^* t$. First, we assume that T is closed. Then, T has the shape

$$\dfrac{\dfrac{T_1}{s_0 \to s_1} \quad \dfrac{\dfrac{T_2}{s_1 \to s_2} \quad \dfrac{\dfrac{T_n}{s_{n-1} \to s_n} \quad \dfrac{}{s_n \to^* t}\,(\mathsf{Refl})}{\vdots}\,(\mathsf{Tran})}{s_1 \to^* t}\,(\mathsf{Tran})}{s \to^* t}\,(\mathsf{Tran})$$

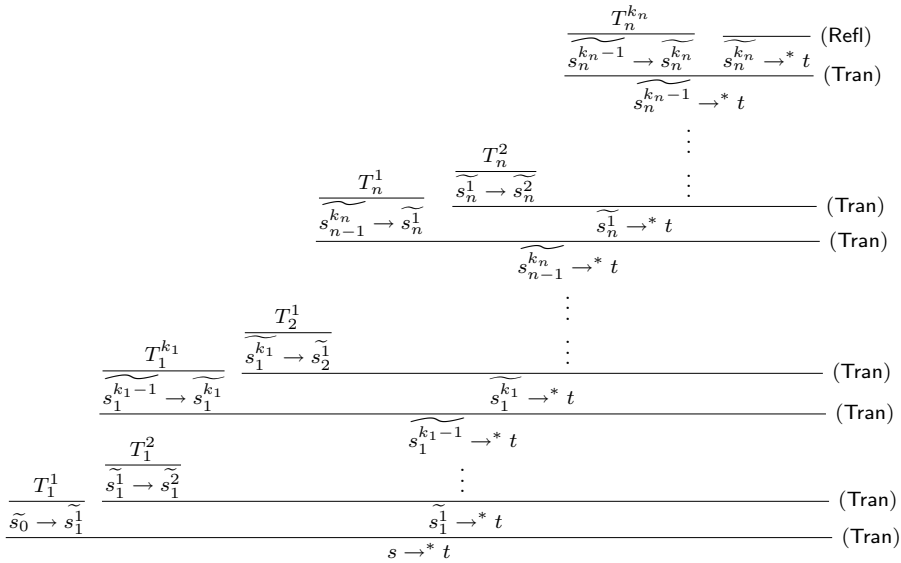
where $s_0 = s$ and $s_n \sim_{\mathcal{E}\cup\mathcal{PA}} t$. By the inductive hypothesis we can assume that each subtree

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

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

$$\dfrac{\dfrac{T_i^1}{s_{i-1} \to s_i^1} \quad \dfrac{\dfrac{T_i^2}{s_i^1 \to s_i^2} \quad \dfrac{\dfrac{T_i^{k_i}}{s_i^{k_i-1} \to s_i^{k_i}} \quad \dfrac{}{s_i^{k_i} \to^* s_i}\,(\mathsf{Refl})}{\vdots}\,(\mathsf{Tran})}{s_i^1 \to^* s_i}\,(\mathsf{Tran})}{s_{i-1} \to^* s_i}\,(\mathsf{Tran})$$

The proof tree $\beta(T)$ is now built by suitably "gluing" the $\beta(U_i)$ together.



If T is not closed since some leftmost T_j^i is not closed, then $\beta(T)$ needs to be cut at the level of T_j^i. 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 $p = \lambda$, i.e., $s \xrightarrow{!}{}^{>\lambda}_{\mathcal{PA}\|\mathcal{E}\setminus\mathcal{S}} \circ \sim_{\mathcal{E}\cup\mathcal{PA}} l\sigma$ 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 4.1. We construct proof trees for $(\mathcal{U}(\mathcal{R}), \mathcal{S}, \mathcal{E})$ with the following head goals:

$$\begin{array}{rcll}
U_n^\rho(t_n, x_n^*)\sigma & \to^* & r\sigma & (G_n)\\
U_n^\rho(s_n, x_n^*)\sigma & \to^* & r\sigma & (H_n)\\
U_{n-1}^\rho(t_{n-1}, x_{n-1}^*)\sigma & \to^* & r\sigma & (G_{n-1})\\
U_{n-1}^\rho(s_{n-1}, x_{n-1}^*)\sigma & \to^* & r\sigma & (H_{n-1})\\
\vdots & & & \\
U_1^\rho(t_1, x_1^*)\sigma & \to^* & r\sigma & (G_1)\\
U_1^\rho(s_1, x_1^*)\sigma & \to^* & r\sigma & (H_1)\\
s & \to^* & t & (K)
\end{array}$$

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

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

$$\dfrac{\dfrac{}{U_n^\rho(t_n, x_n^*)\sigma \to r\sigma}\,(\mathsf{Repl}) \qquad \dfrac{}{r\sigma \to^* r\sigma}\,(\mathsf{Refl})}{U_n^\rho(t_n, x_n^*)\sigma \to^* r\sigma}\,(\mathsf{Tran})$$

using rule (3) from Definition 4.1.

(ii) 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 $(G_k) = U_k^\rho(t_k\sigma, x_k^*\sigma) \to^* r\sigma$. By induction on the tree structure, we can assume that the subtree

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

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

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

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:

$$\dfrac{\dfrac{T_k^1}{u_0' \to u_1'} \qquad \dfrac{\dfrac{T_k^2}{u_1' \to u_2'} \qquad \dfrac{\dfrac{T_k^l}{u_{l-1}' \to u_l'} \qquad \dfrac{T_k}{U_k^\rho(u_l, x_k^*\sigma) \to^* r\sigma}}{u_{l-1}' \to^* r\sigma}\,(\mathsf{Tran})}{\vdots}\,(\mathsf{Tran})}{U_k^\rho(s_k\sigma, x_k^*\sigma) \to^* r\sigma}\,(\mathsf{Tran})$$

where $u'_i = U^\rho_k(u_i, x^*_k\sigma)$.

(iii) 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 $(H_k) = U^\rho_k(s_k\sigma, x^*_k\sigma) \to^* r\sigma$. Then, we can construct a proof tree for the goal $U^\rho_{k-1}(t_{k-1}\sigma, x^*_{k-1}\sigma) \to^* r\sigma$ as follows:

$$\dfrac{\dfrac{}{U^\rho_{k-1}(t_{k-1}\sigma, x^*_{k-1}\sigma) \to U^\rho_k(s_k\sigma, x^*_k\sigma)}\,(\mathsf{Repl}) \qquad T_k}{U^\rho_{k-1}(t_{k-1}\sigma, x^*_{k-1}\sigma) \to^* r\sigma}\,(\mathsf{Tran})$$

where the (Repl) step uses rule (2) from Definition 4.1.

(iv) 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 $(H_1) = U^\rho_1(s_1\sigma, x^*_1\sigma) \to^* r\sigma$. Then, we can construct a proof tree for the goal $s \to^* t$ as follows:

$$\dfrac{\dfrac{}{s \to U^\rho_1(s_1\sigma, x^*_1\sigma)}\,(\mathsf{Repl}) \qquad T_1}{s \to^* t}\,(\mathsf{Tran})$$

where the (Repl) step uses rule (1) from Definition 4.1.

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 $T \subset T'$. □

Lemma 4.3 now easily implies the following result.

Theorem 4.6 *$(\mathcal{R}, \mathcal{S}, \mathcal{E})$ is operationally terminating if $(\mathcal{U}(\mathcal{R}), \mathcal{S}, \mathcal{E})$ is operationally terminating.*

In combination with Lemma 3.6 we get the key result of the present paper.

Corollary 4.7 *$(\mathcal{R}, \mathcal{S}, \mathcal{E})$ is operationally terminating if $(\mathcal{U}(\mathcal{R}), \mathcal{S}, \mathcal{E})$ is terminating.*

Example 4.8 *The following CCES specifies the sieve of Eratosthenes.* $\mathsf{primes}(x)$ *returns a list containing the prime numbers up to x. In this example we have $\mathcal{S} = \mathcal{E} = \emptyset$.*

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

$$\mathsf{isdiv}(x,y) \to^* \mathsf{false} \mid \mathsf{filter}(x,\mathsf{cons}(y,zs)) \to \mathsf{cons}(y,\mathsf{filter}(x,zs))$$

$$\begin{aligned}
\mathsf{isdiv}(x,0) &\to \mathsf{true} && [\![x > 0]\!]\\
\mathsf{isdiv}(x,y) &\to \mathsf{false} && [\![x > y \wedge y > 0]\!]\\
\mathsf{isdiv}(x,x+y) &\to \mathsf{isdiv}(x,y) && [\![x > 0]\!]
\end{aligned}$$

Using Definition 4.1 we obtain the following $\mathcal{U}(\mathcal{R})$.

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

By Corollary 4.7 the CCES $(\mathcal{R},\emptyset,\emptyset)$ is operationally terminating if the unconditional CES $(\mathcal{U}(\mathcal{R}),\emptyset,\emptyset)$ is terminating. Termination of $(\mathcal{U}(\mathcal{R}),\emptyset,\emptyset)$ is shown in [6, Appendix F.2]. ◇

5 Conclusions and Future Work

We have presented conditional constrained equational rewrite systems for specifying algorithms. Rewriting with these systems is based on normalized equational 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 constructor rules and equations. Natural numbers are built-in and constraints are taken from Presburger arithmetic.

We have shown that operational termination of such conditional systems can be reduced to termination of unconditional systems using a syntactic transformation. Powerful methods based on dependency pairs for showing termination of unconditional systems are presented in [9]. These methods can thus be used for showing operational termination of conditional systems as well. Using this approach, oper-

ational termination of several nontrivial conditional systems is shown in [6].

We will next study properties apart from operational termination. In particular, we will investigate confluence and sufficient completeness. Orthogonal to this, we plan to generalize the rewrite relation by considering other built-in theories, most importantly integers instead of natural numbers.

References

[1] Baader, F. and T. Nipkow, "Term Rewriting and All That," Cambridge University Press, 1998.

[2] Bertling, H. and H. Ganzinger, *Completion-time optimization of rewrite-time goal solving*, in: N. Dershowitz, editor, *Proceedings of the 3rd Conference on Rewriting Techniques and Applications (RTA '89)*, Lecture Notes in Computer Science **355** (1989), pp. 45–58.

[3] Blanqui, F., T. Hardin and P. Weis, *On the implementation of construction functions for non-free concrete data types*, in: R. D. Nicola, editor, *Proceedings of the 16th European Symposium on Programming (ESOP '07)*, Lecture Notes in Computer Science **4421** (2007), pp. 95–109.

[4] Clavel, M., F. Durán, S. Eker, Patrick, Lincoln, N. Martí-Oliet, J. Meseguer and C. Talcott, "All About Maude–A High-Performance Logical Framework," Lecture Notes in Computer Science **4350**, Springer-Verlag, 2007.

[5] Durán, F., S. Lucas, C. Marché, J. Meseguer and X. Urbain, *Proving operational termination of membership equational programs*, Higher-Order and Symbolic Computation **21** (2008), pp. 59–88.

[6] Falke, S. and D. 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 at http://www.cs.unm.edu/research/tech-reports/.

[7] Falke, S. and D. Kapur, *Dependency pairs for rewriting with non-free constructors*, in: F. Pfenning, editor, *Proceedings of the 21st Conference on Automated Deduction (CADE '07)*, Lecture Notes in Artificial Intelligence **4603** (2007), pp. 426–442.

[8] Falke, S. and D. Kapur, *Operational termination of conditional rewriting with built-in numbers and semantic data structures*, Technical Report TR-CS-2007-22, Department of Computer Science, University of New Mexico (2007), available at http://www.cs.unm.edu/research/tech-reports/.

[9] Falke, S. and D. Kapur, *Dependency pairs for rewriting with built-in numbers and semantic data structures*, in: A. Voronkov, editor, *Proceedings of the 19th Conference on Rewriting Techniques and Applications (RTA '08)*, Lecture Notes in Computer Science **5117** (2008), pp. 94–109.

[10] Giesl, J. and D. Kapur, *Dependency pairs for equational rewriting*, in: A. Middeldorp, editor, *Proceedings of the 12th Conference on Rewriting Techniques and Applications (RTA '01)*, Lecture Notes in Computer Science **2051** (2001), pp. 93–108.

[11] Lucas, S., C. Marché and J. Meseguer, *Operational termination of conditional term rewriting systems*, Information Processing Letters **95** (2005), pp. 446–453.

[12] Marché, C., *Normalized rewriting: An alternative to rewriting modulo a set of equations*, Journal of Symbolic Computation **21** (1996), pp. 253–288.

[13] Ohlebusch, E., "Advanced Topics in Term Rewriting," Springer-Verlag, 2002.

[14] Peterson, G. E. and M. E. Stickel, *Complete sets of reductions for some equational theories*, Journal of the ACM **28** (1981), pp. 233–264.

[15] Presburger, M., *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*, 1929, pp. 92–101.