Applications and extensions of context-sensitive rewriting☆

Salvador Lucas[1]

[a] *DSIC & VRAIN, Universitat Politècnica de València, Spain*
http://slucas.webs.upv.es/

Abstract

Context-sensitive rewriting is a restriction of term rewriting which is obtained by imposing replacement restrictions on the arguments of function symbols. It has proven useful to analyze computational properties of programs written in sophisticated rewriting-based programming languages such as CafeOBJ, Haskell, Maude, OBJ*, etc. Also, a number of *extensions* (e.g., to *conditional rewriting* or *constrained equational systems*) and *generalizations* (e.g., *controlled rewriting* or *forbidden patterns*) of context-sensitive rewriting have been proposed. In this paper, we provide an overview of these applications and related issues.

Keywords: Program Analysis, Programming Languages, Term Rewriting

---

1. Introduction

When computing with reduction-based systems, *rules cannot be applied just anywhere* [98, page 34]. For instance, in *Generalized Rewrite Theories* [16, 17], the use of *replacement restrictions*, aimed at avoiding this, brings "*a substantial increase in expressive power of the Rewriting Logic formalism*" (see [94]) that

> "has to do with the fact that rewrites *should not happen everywhere*, because in many applications suitable evaluation strategies or context-dependent rewrites could considerably improve performance and even avoid non-termination. Correspondingly, rewrite theories can be generalized by *forbidding rewriting under certain operators or operator positions* (frozen operators and arguments). Although this could be regarded as a purely *operational aspect*, the frequent need for it in many applications suggests that it should be supported directly at the semantic level of rewrite theories." [17, Section 1]

Perhaps this observation explains why replacement restrictions imposed by *context-sensitive rewriting* (*CSR* [79, 86]) have been used to analyze semantic

☆Partially supported by the EU (FEDER), and projects RTI2018-094403-B-C32 and PROMETEO/2019/098.

```
mod ExSec11_1_Luc02 is
 sort S .
 ops 0 nil : -> S .           ops dbl s recip sqr terms : S -> S .
 ops add first : S S -> S .   op _:_ : S S -> S [frozen (2)] .
 vars m n x : S .             var xs : S .
 rl add(0,n) => n .       rl add(s(m),n) => s(add(m,n)) .
 rl dbl(0) => 0 .         rl dbl(s(n)) => s(s(dbl(n))) .
 rl sqr(0) => 0 .         rl sqr(s(n)) => s(add(sqr(n),dbl(n))) .
 rl first(0,xs) => nil . rl first(s(n),x : xs) => x : first(n,xs) .
 rl terms(n) => recip(sqr(n)) : terms(s(n)) .
endm
```

Figure 1: Maude specification to approximate $\pi^2/6$

aspects and properties of several programming languages and systems. Reporting on these *applications* of *CSR* is the main purpose of this paper.

In *CSR* a *replacement map* μ specifies, for each k-ary symbol f, the *active* argument positions $\mu(f) \subseteq \{1, \ldots, k\}$ where rewriting is allowed in a function call $f(t_1, \ldots, t_k)$ (while any other argument t_i with $i \notin \mu(f)$ remains *frozen*). In unrestricted rewriting, if a term s rewrites into t, then, for all k-ary function symbols f and arguments i, $1 \leq i \leq k$, we have that $f(\cdots \underbrace{s}_{i} \ldots)$ rewrites into $f(\cdots \underbrace{t}_{i} \ldots)$, i.e., s still rewrites into t when surrounded by any syntactic context $f(\cdots \underbrace{\quad}_{i} \ldots)$. In *CSR*, this happens (and it is top-down propagated) for indices $i \in \mu(f)$ only.

A motivating example. In connection with the use of *CSR* to *reinforce* termination of programs, consider the Maude [19] program in Figure 1 which is a Maude presentation of the TRS in [82, Example 2] with the replacement map used in [86, Examples 6.7 and 9.8] and that can be used to compute approximations to $\pi^2/6 = 1,64493406684823\cdots$ as the sum of the n components of an initial sublist $s = \mathtt{first}(\mathtt{s}^{\mathtt{n}}(0), \mathtt{terms}(\mathtt{s}(0)))$[1] of the infinite list `terms(s(0))`, consisting of $\frac{1}{1^2}, \frac{1}{2^2}, \frac{1}{3^2}, \ldots, \frac{1}{n^2}, \ldots$, where each reciprocal fraction $\frac{1}{m}$ above is represented as $\mathtt{recip}(\mathtt{s}^{\mathtt{m}}(0))$ see [44, page 265]. Note that, if we drop the *frozenness* annotation for the list constructor `_:_`, i.e.,

```
op _:_ : S S -> S [frozen (2)] .
```

which corresponds to a replacement map μ given by $\mu(\mathtt{\_:\_}) = \{1\}$ and $\mu(f) = \{1, \ldots, k\}$ for any other k-ary symbol f, the program is *not* terminating due to

[1] Here and in the following, for the sake of readability, we often use $\mathtt{s}^{\mathtt{n}}(0)$ instead of $\underbrace{\mathtt{s}(\cdots(\mathtt{s}(}_{n}\, 0\, \underbrace{)\cdots)}_{n}$.

the last program rule which permits an infinite recursion on successive recursive calls to $\mathtt{terms}(\mathtt{s}^n(0))$ for $n \geq 0$. The attempt to use the Maude command `rewrite` to obtain the first 4 components of the sequence approximating $\pi^2/6$ yields:

```
Maude> rew first(s(s(s(s(0)))),terms(s(0))) .
rewrite in ExSec11_1_Luc02 : first(s(s(s(s(0)))), terms(s(0))) .
rewrites: 6 in 0ms cpu (0ms real) (750000 rewrites/second)
result S: recip(s(0)) : first(s(s(s(0))), terms(s(s(0))))
```

The replacement restrictions make the program terminating by avoiding reductions on the second argument of `_:_` (thus cutting the aforementioned infinite recursion). However, the program *fails* to obtain the expected outcome $[\frac{1}{1}, \frac{1}{4}, \frac{1}{9}, \frac{1}{16}]$, represented by the program expression

$$\mathtt{recip}(\mathtt{s}(0)) : \mathtt{recip}(\mathtt{s}^4(0)) : \mathtt{recip}(\mathtt{s}^9(0)) : (\mathtt{recip}(\mathtt{s}^{16}(0)) : \mathtt{nil}.$$

In Section 10.2, though, we show how to overcome this problem.

Contributions of the paper. In [86] the basic aspects and facts about *CSR* were described, including the practical use of *CSR* by means of the interpreters of existing rewriting-based languages with capabilities to express replacement maps (in particular, Maude, as exemplified above), the ability of *CSR* to simulate unrestricted rewriting, the analysis of confluence and termination of *CSR*, and how *CSR* can be used to compute canonical forms (head-normal forms, values, normal forms, and (approximations of) infinite normal forms) which are of interest in rewriting-based computations and (algebraic, equational, and functional) programming languages.

In this paper we focus on *applications* and *extensions* of *CSR* developed in the last 20 years by several authors to model, investigate, and prove properties of rewriting and rewriting-based programming languages. After some preliminary definitions in Sections 2 and 3 (which try to make this paper sufficiently self-contained), we explore the use of *CSR* to analyze termination properties of variants of rewriting like first-order lazy functional programs, and *innermost* and *outermost* rewriting (Section 4), conditional rewriting (Section 5), productivity (Section 6), and runtime complexity (Section 7). Section 8 investigates the interaction of *CSR* with related notions of rewriting, like *conditional* rewriting, *constrained* rewriting, *equational* rewriting, and *narrowing*. Section 9 shows how the notion of *CSR* has been modified to be used with rewriting to achieve more flexibility, leading to *on-demand strategy annotations*, *lazy* rewriting, rewriting with *forbidden patterns*, and *controlled* term rewriting. Section 10 shows how the theory of *CSR* has been used to analyze properties of OBJ programs[2] and improve their computations by including (in Maude) techniques developed for *CSR* like normalization via μ-normalization which can now be used through

[2]As in [54], by OBJ we mean OBJ2, OBJ3, CafeOBJ, or Maude.

Maude's strategy language. In the development of these sections, we made an effort to provide a uniform presentation and draw connections among them and provide illustrative examples of use. Section 11 concludes. Some technical results in the paper are new (thus labeled with $^{(\star)}$).

2. Preliminaries

This section collects some definitions and notations from term rewriting. More details and missing notions can be found in [12, 116]. In the following, $\mathcal{P}(A)$ denotes the powerset of a set A. Given a binary relation $\mathsf{R} \subseteq A \times A$ on a set A, we denote its *transitive* closure by R^+, and its *reflexive and transitive* closure by R^*. An element $a \in A$ is *irreducible* (or an R-*normal form*), if there exists no b such that $a \;\mathsf{R}\; b$. We say that b is an R-normal form of a (written $a \,\mathsf{R}^!\, b$), if $a \;\mathsf{R}^* b$ and b is an R-normal form. We also say that a is R-normalizing, i.e., a has an R-normal form. Also, R is *normalizing* if every $a \in A$ has an R-normal form. Given $a \in A$, if there is no infinite sequence $a = a_1 \;\mathsf{R}\; a_2 \;\mathsf{R}\; \cdots \;\mathsf{R}\; a_n \;\mathsf{R} \cdots$, then a is R-*terminating* (or *well-founded*[3]); R is *terminating* if a is R-terminating for all $a \in A$. We say that R is *confluent* if, for every $a, b, c \in A$, whenever $a \;\mathsf{R}^* b$ and $a \;\mathsf{R}^* c$, there exists $d \in A$ such that $b \;\mathsf{R}^* d$ and $c \;\mathsf{R}^* d$.

Throughout the paper, $\mathcal{X}$ denotes a countable set of variables and $\mathcal{F}$ denotes a signature, i.e., a set of function symbols $f, g \ldots$, each having a fixed arity $ar(f)$. The set of terms built from $\mathcal{F}$ and $\mathcal{X}$ is $\mathcal{T}(\mathcal{F}, \mathcal{X})$. The set of variables in a term t is denoted $\mathcal{V}ar(t)$. A term is said to be *linear* if it has no multiple occurrences of a single variable. Terms are viewed as labelled trees in the usual way. *Positions* $p, q, \ldots$ are represented by chains of positive natural numbers used to address subterms of t. Given positions p, q, we denote their concatenation as $p.q$. Positions are ordered by the standard prefix ordering $\leq$. Given a set of positions P, $minimal_{\leq}(P)$ is the set of minimal positions of P w.r.t. $\leq$. If p is a position, and Q is a set of positions, $p.Q = \{p.q \mid q \in Q\}$. We denote the empty chain by Λ. The set of positions of a term t is denoted $\mathcal{P}os(t)$. Positions of non-variable symbols in t are denoted as $\mathcal{P}os_{\mathcal{F}}(t)$, and $\mathcal{P}os_{\mathcal{X}}(t)$ are the positions of variables. The subterm of t at position p is denoted as $t|_p$ and $t[s]_p$ is the term t with the subterm at position p replaced by s. The symbol labelling the root of t is denoted as $root(t)$. Given terms t and s, $\mathcal{P}os_s(t)$ denotes the set of positions of the subterm s in t, i.e., $\mathcal{P}os_s(t) = \{p \in \mathcal{P}os(t) \mid t|_p = s\}$.

Two terms s and t *unify* if there is a substitution σ (i.e., a *unifier*) such that $\sigma(s) = \sigma(t)$. If s and t unify, then there is a (unique up to variable renaming) *most general unifier* (*mgu*) θ of s and t satisfying that, for any other unifier σ of s and t, there is a substitution τ such that, for all $x \in \mathcal{X}$, $\sigma(x) = \tau(\theta(x))$.

A rewrite rule is an ordered pair (ℓ, r), written $\ell \to r$, with $\ell, r \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, $\ell \notin \mathcal{X}$ and $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(\ell)$. The left-hand side (*lhs*) of the rule is ℓ and r is the right-hand side (*rhs*). A TRS is a pair $\mathcal{R} = (\mathcal{F}, R)$ where R is a set of

[3] See [116, Definition 2.1.1] and the paragraph below this definition for a clarifying discussion about the use of 'well-founded' and 'terminating' in Mathematics and Computer Science.

rewrite rules. $L(\mathcal{R})$ denotes the set of *lhs*'s of $\mathcal{R}$. An instance $\sigma(\ell)$ of a *lhs* ℓ of a rule is a redex. A TRS $\mathcal{R}$ is left-linear if for all $\ell \in L(\mathcal{R})$, ℓ is a linear term. Given $\mathcal{R} = (\mathcal{F}, R)$, we consider $\mathcal{F}$ as the disjoint union $\mathcal{F} = \mathcal{C} \uplus \mathcal{D}$ of symbols $c \in \mathcal{C}$, called *constructors* and symbols $f \in \mathcal{D}$, called *defined functions*, where $\mathcal{D} = \{root(\ell) \mid \ell \to r \in R\}$ and $\mathcal{C} = \mathcal{F} - \mathcal{D}$. We often denote as $\mathcal{C}_\mathcal{R}$ (resp. $\mathcal{D}_\mathcal{R}$) the constructor (resp. defined) symbols of $\mathcal{R}$. Then, $\mathcal{T}(\mathcal{C}, \mathcal{X})$ (resp. $\mathcal{T}(\mathcal{C})$) is the set of (ground) constructor terms.

A term $s \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ rewrites to t (at position p), written $s \stackrel{p}{\to}_\mathcal{R} t$ (or just $s \to t$), if $s|_p = \sigma(\ell)$ and $t = s[\sigma(r)]_p$, for some rule $\rho : \ell \to r \in R$, $p \in \mathcal{P}os(s)$ and substitution σ. A TRS is confluent (terminating) if $\to$ is confluent (terminating). A term s is root-stable (or a head-normal form) if there is no redex t such that $s \to^* t$. A term is said to be root-normalizing if it has a root-stable reduct. A term is said to be normalizing if it is $\to$-normalizing.

Two rules $\ell \to r$ and $\ell' \to r'$ such that $\mathcal{V}ar(\ell) \cap \mathcal{V}ar(\ell') = \varnothing$ (rename the variables if necessary) define a *critical pair* $\langle \sigma(\ell)[\sigma(r')]_p, \sigma(r)\rangle$ if $p \in \mathcal{P}os_\mathcal{F}(\ell)$ (the *critical* position) is a *nonvariable position* of ℓ such that $\ell|_p$ and ℓ' *unify* with *mgu* σ. The case $\ell \to r = \ell' \to r'$ (up to renaming) and $p = \Lambda$ is *excluded*. A left-linear TRS without critical pairs is called *orthogonal*.

3. Context-sensitive rewriting

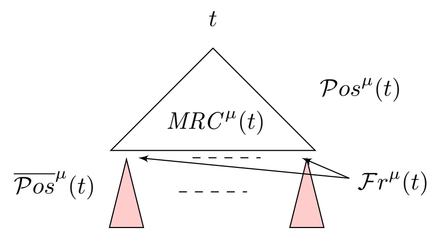
The concepts and notations in this section are extracted from [86]. A *replacement map* is a mapping $\mu : \mathcal{F} \to \mathcal{P}(\mathbb{N})$ satisfying that, for all symbols f in $\mathcal{F}$, $\mu(f) \subseteq \{1, \ldots, ar(f)\}$. The set of replacement maps for a signature $\mathcal{F}$ is $M_\mathcal{F}$ (for TRSs $\mathcal{R} = (\mathcal{F}, R)$, we use $M_\mathcal{R}$ instead). Replacement maps are compared as follows: $\mu \sqsubseteq \mu'$ if for all $f \in \mathcal{F}$, $\mu(f) \subseteq \mu'(f)$; we often say that μ is *more restrictive* than μ'. Extreme cases are $\mu_\bot$, which disallows replacements in all arguments: $\mu_\bot(f) = \varnothing$ for all $f \in \mathcal{F}$; and $\mu_\top$, which restricts no replacement: $\mu_\top(f) = \{1, \ldots, k\}$ for all k-ary symbols $f \in \mathcal{F}$. We say that a binary relation R on terms is *μ-monotonic* if for all k-ary symbols f, $i \in \mu(f)$, and terms $s_1, \ldots, s_k, t_i$, if $s_i \,\mathsf{R}\, t_i$, then $f(s_1, \ldots, s_i, \ldots, s_k) \,\mathsf{R}\, f(s_1, \ldots, t_i, \ldots, s_k)$. We say that R is *monotonic* if it is $\mu_\top$-monotonic.

Active and frozen positions. The replacement restrictions introduced by μ on the *arguments* of function *symbols* are lifted to *positions* of *terms* $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$: the set $\mathcal{P}os^\mu(t)$ of *μ-replacing* or *active* positions of t is:

$$\mathcal{P}os^\mu(t) = \begin{cases} \{\Lambda\} & \text{if } t \in \mathcal{X} \\ \{\Lambda\} \cup \bigcup_{i \in \mu(f)} i.\mathcal{P}os^\mu(t_i) & \text{if } t = f(t_1, \ldots, t_k) \end{cases}$$

The set $\overline{\mathcal{P}os^\mu}(t) = \mathcal{P}os(t) - \mathcal{P}os^\mu(t)$ contains the *non-μ-replacing* or *frozen* positions of t.

The frozen positions of term t (depicted in *red* in the joint diagram) have a *frontier* set $\mathcal{F}r^{\mu}(t) = minimal_{\leq}(\overline{\mathcal{P}os^{\mu}}(t))$ with the active positions. The *maximal replacing context* $MRC^{\mu}(t) = t[\Box]_{\mathcal{F}r^{\mu}(t)}$ of t is the prefix of t whose positions are active. Hence, t can be written $t = C[t_1, \ldots, t_n]$ with $C = MRC^{\mu}(t)$ with the frozen positions $\mathcal{F}r^{\mu}(t)$ filled with appropriate terms $t_1, \ldots, t_n$.



Context-sensitive rewriting. *CSR* is the restriction of rewriting obtained when a replacement map μ is used to specify the redex positions that can be contracted.

Definition 1 (Context-sensitive rewriting). *Let $\mathcal{R}$ be a TRS, $\mu \in M_{\mathcal{R}}$ and s and t be terms. Then, s μ-rewrites to t, written $s \overset{p}{\hookrightarrow}_{\mathcal{R},\mu} t$ (or $s \hookrightarrow_{\mathcal{R},\mu} t$, also $s \hookrightarrow_{\mu} t$, or even $s \hookrightarrow t$), if $s \overset{p}{\to}_{\mathcal{R}} t$ and p is active in s (i.e., $p \in \mathcal{P}os^{\mu}(s)$).*

Terms t which cannot be μ-rewritten are called *μ-normal forms*. In the following $\mathsf{NF}^{\mu}_{\mathcal{R}}$ denotes the set of μ-normal forms of $\mathcal{R}$.

Example 1. *The μ-rewriting sequence corresponding to the* Maude *evaluation in the introduction is the following:*

$$\begin{array}{rl}
\mathsf{first}(\mathsf{s}^4(0)), \underline{\mathsf{terms}(\mathsf{s}(0))}) & \hookrightarrow_{\mu} \underline{\mathsf{first}(\mathsf{s}^4(0)), \mathsf{recip}(\mathsf{sqr}(\mathsf{s}(0)))) : \mathsf{terms}(\mathsf{s}^2(0)))} \\
& \hookrightarrow_{\mu} \mathsf{recip}(\underline{\mathsf{sqr}(\mathsf{s}(0)))}) : \mathsf{first}(\mathsf{s}^3(0)), \mathsf{terms}(\mathsf{s}^2(0))) \\
& \hookrightarrow_{\mu} \mathsf{recip}(\mathsf{s}(\underline{\mathsf{sqr}(0)} + \mathsf{dbl}(0))) : \mathsf{first}(\mathsf{s}^3(0)), \mathsf{terms}(\mathsf{s}^2(0))) \\
& \hookrightarrow_{\mu} \mathsf{recip}(\mathsf{s}(0 + \underline{\mathsf{dbl}(0)})) : \mathsf{first}(\mathsf{s}^3(0)), \mathsf{terms}(\mathsf{s}^2(0))) \\
& \hookrightarrow_{\mu} \mathsf{recip}(\mathsf{s}(\underline{0 + 0})) : \mathsf{first}(\mathsf{s}^3(0)), \mathsf{terms}(\mathsf{s}^2(0))) \\
& \hookrightarrow_{\mu} \mathsf{recip}(\mathsf{s}(0)) : \mathsf{first}(\mathsf{s}^3(0)), \mathsf{terms}(\mathsf{s}^2(0)))
\end{array}$$

which stops in the μ-normal form $t =$ $\mathsf{recip}(\mathsf{s}(0)) : \mathsf{first}(\mathsf{s}(\mathsf{s}(\mathsf{s}(0))), \mathsf{terms}(\mathsf{s}(0)))$ which displays the first component $\frac{1}{1}$ (denoted $\mathsf{recip}(\mathsf{s}(0))$*) of the sequence only.*

A pair $(\mathcal{R}, \mu)$ where $\mathcal{R}$ is a TRS and $\mu \in M_{\mathcal{R}}$ is often called a CS-TRS. A TRS $\mathcal{R}$ is *μ-terminating* if $\hookrightarrow_{\mu}$ is terminating. Several tools have been furnished with capabilities to automatically prove termination of *CSR*, see [86, Section 7.2]. In this paper, we often use MU-TERM [60]:

`http://zenon.dsic.upv.es/muterm/`

Inference system and theory for a context-sensitive rewrite system. An alternative definition of *CSR*, which is used in Section 8, relies on the notion of provability with a given inference system. Consider the inference rules in Figure 2, where $(\mathrm{C})_{f,i}$ is parametric on a function symbol f and an argument $1 \leq i \leq ar(f)$, and $(\mathrm{Rl})_{\ell \to r}$ is parametric on a rule $\ell \to r$. The inference system

$$\mathfrak{I}_{CSR}[\mathbb{S}, \mathbb{M}, \mathbb{R}] = \{(\mathrm{Rf}), (\mathrm{T})\} \cup \{(\mathrm{C})_{f,i} \mid f \in \mathbb{S}, i \in \mathbb{M}(f)\} \cup \{(\mathrm{Rl})_{\ell \to r} \mid \ell \to r \in \mathbb{R}\}$$

$$
\begin{array}{llll}
\text{(Rf)} & \dfrac{}{x \to^* x} & \text{(C)}_{f,i} & \dfrac{x_i \to y_i}{f(x_1,\ldots,x_i,\ldots,x_k) \to f(x_1,\ldots,y_i,\ldots,x_k)} \\
\text{(T)} & \dfrac{x \to y \quad y \to^* z}{x \to^* z} & \text{(Rl)}_{\ell\to r} & \dfrac{}{\ell \to r}
\end{array}
$$

Figure 2: Some parametric inference rules

is parametric on signatures (parameter $\mathbb{S}$), replacement maps (parameter $\mathbb{M}$), and TRSs (parameter $\mathbb{R}$). Given $\mathcal{R} = (\mathcal{F}, R)$ and $\mu \in M_{\mathcal{R}}$, let $\mathcal{I}(\mathcal{R},\mu)$ be the specific inference system obtained from $\mathfrak{I}_{CSR}$ by instantiating all parameters:

$$\mathcal{I}(\mathcal{R},\mu) \quad = \quad \mathfrak{I}_{CSR}[\mathcal{F},\mu,\mathcal{R}]$$

Still, rules in $\mathcal{I}(\mathcal{R},\mu)$ are *schematic*: each inference rule $\frac{B_1 \ \cdots \ B_n}{A}$ can be used under any *instance* $\frac{\sigma(B_1) \ \cdots \ \sigma(B_n)}{\sigma(A)}$ of the rule by a substitution σ. Now, for all terms s, t, we have $s \hookrightarrow_{\mathcal{R},\mu} t$ (resp. $s \hookrightarrow^*_{\mathcal{R},\mu} t$) iff $s \to t$ (resp. $s \to^* t$) can be proved in $\mathcal{I}(\mathcal{R},\mu)$. We obtain a first-order theory $\overline{\mathcal{R}^\mu}$ from $\mathcal{I}(\mathcal{R},\mu)$ by translating each inference rule $\frac{B_1 \ \cdots \ B_n}{A}$ into a universally quantified formula $(\forall \vec{x}) B_1 \wedge \cdots \wedge B_n \Rightarrow A$, where $\vec{x} = x_1, \ldots, x_m$ are the variables occurring in $A, B_1, \ldots, B_n$.

Remark 1. *Unrestricted rewriting can be viewed as a particular case of* CSR *where the replacement map* $\mu_\top$ *is used. We let* $\mathfrak{I}_{TRS}[\mathbb{S},\mathbb{R}] = \mathfrak{I}_{CSR}[\mathbb{S},\mu_\top,\mathbb{R}]$.

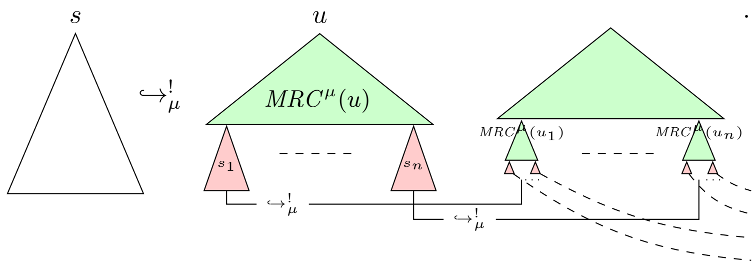
Canonical context-sensitive rewriting. The *canonical replacement map* $\mu^{can}_{\mathcal{R}}$ of a TRS $\mathcal{R}$ is *the most restrictive replacement map* μ *ensuring that the non-variable subterms of the left-hand sides* ℓ *of the rules* $\ell \to r$ *of* $\mathcal{R}$ *are all active*, i.e., $\mathcal{P}os_{\mathcal{F}}(\ell) \subseteq \mathcal{P}os^\mu(\ell)$: for each symbol $f \in \mathcal{F}$ and $i \in \{1, \ldots, ar(f)\}$,

$$i \in \mu^{can}_{\mathcal{R}}(f) \quad \text{iff} \quad \exists \ell \to r \in \mathcal{R}, p \in \mathcal{P}os_{\mathcal{F}}(\ell), (root(\ell|_p) = f \wedge p.i \in \mathcal{P}os_{\mathcal{F}}(\ell)).$$

Given a TRS $\mathcal{R}$, we let $CM_{\mathcal{R}} = \{\mu \in M_{\mathcal{R}} \mid \mu^{can}_{\mathcal{R}} \sqsubseteq \mu\}$ be the set of replacement maps that are *less (or equally) restrictive* than the canonical replacement map. If $\mu \in CM_{\mathcal{R}}$, we also say that μ is *a* canonical replacement map for $\mathcal{R}$; if μ is exactly $\mu^{can}_{\mathcal{R}}$, we will speak about *the* canonical replacement map of $\mathcal{R}$.

For TRSs $\mathcal{R}$ and $\mu \in CM_{\mathcal{R}}$, we often say that $\hookrightarrow_{\mathcal{R},\mu}$ performs *canonical* CSR [82]. Canonical *CSR* is useful in head-normalization, normalization, and infinitary normalization with left-linear TRSs. In particular, the *normalization-via-μ-normalization* procedure $\mathtt{norm}_\mu$ in Figure 3 permits the *layered* normalization of expressions using *CSR* by successive steps of μ-normalization of the maximal frozen subterms of the μ-normal forms which are obtained in the previous layer, see [86, Section 9] for further motivation and comparisons with well-known normalization procedures in functional programming or term rewriting like, e.g., normalization via root-stabilization [96].

Frozen arguments in Maude *for a practical use of context-sensitive rewriting.* When dealing with Maude *system modules* [19, Chapter 6], each k-ary function



$$\begin{array}{l}
\mathtt{norm}_\mu(s) = C[\mathtt{norm}_\mu(s_1), \ldots, \mathtt{norm}_\mu(s_n)] \\
\textbf{where} \\
\quad u \text{ is a } \mu\text{-normal form of } s \\
\quad u = C[s_1, \ldots, s_n] \text{ for } C[, \ldots,] = MRC^\mu(u)
\end{array}$$

Figure 3: Normalization via μ-normalization [86, Section 9.3]

symbol f has a set $\varphi(f) \subseteq \{1, \ldots, k\}$ of *frozen* argument positions that *cannot* be rewritten [22, Section 3]. Dually, φ restricts rewritings as a replacement map μ_φ given by $\mu_\varphi(f) = \{1, \ldots, ar(f)\} - \varphi(f)$ for all symbols f.

Example 2. *Program* `ExSec11_1_Luc02` *specifies* $\varphi(:) = \{2\}$ *and* $\varphi(f) = \varnothing$ *for any other symbol* f*, i.e., only the second argument of* `_:_` *is frozen. Accordingly,* $\mu_\varphi(:) = \{1\}$ *and* $\mu_\varphi(f) = \{1, \ldots, k\}$ *for any other* k*-ary symbol* f*.*

Section 10.2 describes an implementation of normalization via μ-normalization for Maude. We use it to obtain the normal form of s in the introduction.

4. Termination of *CSR* and other termination properties

In the following, we show how termination of *CSR* has been used to prove other termination properties like termination of variants of rewriting with TRSs like in lazy (first-order) functional programs (Section 4.1), innermost rewriting (Section 4.2), and outermost rewriting (Section 4.3).

4.1. Termination of lazy functional programs

CSR can be used to model *non-strict* evaluation ($\overset{\mathsf{ns}}{\to}$) in TRSs and first-order functional languages [49, Section 2].

Definition 2 (non-strict evaluation). *[49, Definition 2] Let* $\mathcal{R}$ *be a left-linear TRS. A term* s *rewrites to a term* t *with* non-strict evaluation *(written* $s \overset{\mathsf{ns}}{\to} t$*) iff there is a rule* $\ell \to r$ *such that* $root(s) = root(\ell)$ *and either (i)* $s = \sigma(\ell)$ *and* $t = \sigma(r)$ *for some substitution* σ*, or (ii)* $s|_p \overset{\mathsf{ns}}{\to} t'$ *and* $t = s[t']_p$ *for the minimum position* $p \in \mathcal{P}os_{\mathcal{F}}(\ell) \cap \mathcal{P}os(s)$ *with respect to the lexicographical order on positions such that* $root(s|_p) \neq root(\ell|_p)$*.*

Non-strict evaluation mimics the evaluation strategy of *first-order*, *unconditional* programs in lazy functional languages like Haskell [69]. Such programs can be viewed as left-linear TRSs. Giesl and Middeldorp prove the following.

Theorem 1. *[49, Theorem 3] Let $\mathcal{R}$ be a left-linear TRS. If $\mathcal{R}$ is $\mu^{can}_{\mathcal{R}}$-terminating, then $\stackrel{\mathsf{ns}}{\to}$ is terminating.*

Example 3. *Consider the* Haskell *program* `Nats` *for $\mathcal{R}$ in [49, Sect. 1]:*

```
data Nat = Z | S Nat deriving Show

nats = adx zeros
adx (x : y) = incr (x : adx y)
zeros = Z : zeros
hd (x : y) = x
tl (x : y) = y
incr (x : y) = (S x) : incr y
```

with $\mu = \mu^{can}_{\mathcal{R}}$, i.e., $\mu(:) = \mu(\mathsf{s}) = \varnothing$ and $\mu(\mathsf{adx}) = \mu(\mathsf{hd}) = \mu(\mathsf{incr}) = \mu(\mathsf{tl}) = \{1\}$. Although `Nats` *is nonterminating (due to the third rule), it is μ-terminating (use* MU-TERM*). By Theorem 1, $\stackrel{\mathsf{ns}}{\to}$ is terminating.*

Giesl and Middeldorp argue that $\mu^{can}_{\mathcal{R}}$-termination is

> "a sufficient but not a necessary criterion for the termination of non-strict evaluation (and hence of the corresponding functional program)" [49, page 385].

However, from termination of $\stackrel{\mathsf{ns}}{\to}$ for (the TRS associated to) `Nats`, for instance, we *cannot* conclude termination of `Nats` in the usual sense, i.e., the absence of infinite evaluation sequences. For instance, the evaluation of `zeros` in `Nats` leads to an infinite sequence $\mathtt{Z},\ldots,\mathtt{Z},\ldots$ computed and steadily displayed (unless broken by the user) by the Haskell interpreter:

```
*Main> zeros
[Z,Z,Z,Z,Z,Z,Z,Z,Z,Z,Z,Z,Z,Z,Z,Z,Z,Z,Z,Z,Z,Z, ...,Z,Interrupted.
*Main>
```

This is not surprising, as in lazy functional programming (e.g., in Haskell) termination in the usual sense is often 'neglected', as the ability to deal with, or even *approximate*, infinite values (like the infinite list of `Z`'s denoted by `zeros`) is an asset. Of course, the ability to *obtain a value when possible* is also an asset, but this fits the notion of *normalization* rather than termination.

CSR is a *restriction of rewriting* rather than a rewriting strategy. Hence, it is *not* forced to reduce beyond μ-normal forms. Mappings H returning a nonempty set $\mathsf{H}(t) \subseteq \mathcal{P}os^{\mu}(t)$ of positions of active redexes for each term t which is *not* a μ-normal form are called (one-step) *context-sensitive rewriting strategies* (or CS strategies) [82, Definition 2]. We also speak of μ-strategies to make the specific replacement map μ explicit. We write $s \hookrightarrow_{\mathsf{H}} t$ if $s \stackrel{p}{\hookrightarrow}_{\mu} t$ for some $p \in \mathsf{H}(s)$. We can use (one-step) CS strategies H to obtain a strategy S_{H} in the usual sense, i.e., always reducing terms unless they are normal forms [82, Section 5]:

$$\mathsf{S}_{\mathsf{H}}(t) = \begin{cases} \mathsf{H}(t) & \text{if } t \notin \mathsf{NF}^{\mu}_{\mathcal{R}} \\ \cup_{1 \leq i \leq n} p_i.\mathsf{S}_{\mathsf{H}}(t_i) & \text{otherwise, where:} \\ & \quad C[\,] = MRC^{\mu}(t), t = C[t_1, \ldots, t_n], \\ & \quad \text{and } t_i = t|_{p_i} \text{ for } 1 \leq i \leq n \end{cases} \tag{1}$$

Corollary 1. *[82, Corollary 10] Let $\mathcal{R}$ be a left-linear and confluent TRS, and $\mu \in CM_{\mathcal{R}}$. If $\mathcal{R}$ is μ-terminating, then S_{H} is a normalizing strategy for every μ-strategy H.*

Thus, for left-linear, confluent, and μ-terminating TRSs $\mathcal{R}$ (with $\mu \in CM_{\mathcal{R}}$), we obtain a normalizing strategy S_{H} from *any* μ-strategy H. In this way, Corollary 1 provides a more realistic formulation of the use of termination of canonical *CSR* in lazy functional languages when dealing with first-order, unconditional and confluent programs.

Example 4. *The TRS $\mathcal{R}$ corresponding to* `Nats` *is left-linear and orthogonal (hence confluent, see, e.g., [12]). By Corollary 1, the evaluation of every normalizing initial expression e by using S_{H} for an arbitrary μ-strategy H with $\mu \in CM_{\mathcal{R}}$ always finishes.*

4.2. Termination of innermost rewriting

In *innermost* rewriting computations (written $s \to_{\mathsf{i}} t$), rewriting steps contract innermost redexes of s, i.e., those which contain no other redex. Whenever a rule $\ell \to r$ is used to perform an innermost rewriting step $s \to_{\mathsf{i}} t$, with $s|_p = \sigma(\ell)$ and $t = s[\sigma(r)]_p$, the matching substitution σ is *normalized*, i.e., for all $x \in \mathcal{V}ar(\ell)$, $\sigma(x)$ is a normal form. Thus, we can restrict reductions on the arguments of function symbols f in r with a replacement map μ so that for all $p \notin Pos^{\mu}(r)$, $r|_p$ is a *constructor subterm* of r. In this way, $\sigma(r|_p)$ is a normal form, where no rewriting can be performed anyway. Hence, the *usable arguments* $i \in \mu^{\mathsf{UA}}_{\mathcal{R}}(f)$ for a k-ary symbol f are those satisfying that there is a subterm $f(t_1, \ldots, t_i, \ldots, t_k)$ of the rhs r of a rule $\ell \to r \in \mathcal{R}$ such that t_i contains a *defined* symbol. We have the following.

Theorem 2. *[39, Corollary 11] A TRS $\mathcal{R}$ is innermost terminating if $\mathcal{R}$ is $\mu^{\mathsf{UA}}_{\mathcal{R}}$-terminating.*

Example 5. *Let $\mathcal{R}$ be the following nonterminating TRS (Toyama's example):*

$$\mathsf{c} \to \mathsf{a} \qquad \mathsf{c} \to \mathsf{b} \qquad \mathsf{f}(\mathsf{a}, \mathsf{b}, x) \to \mathsf{f}(x, x, x)$$

Note that $\mu^{\mathsf{UA}}_{\mathcal{R}}(\mathsf{f}) = \varnothing$. The $\mu^{\mathsf{UA}}_{\mathcal{R}}$-termination of $\mathcal{R}$ can be proved with MU-TERM*. By Theorem 2, innermost termination of $\mathcal{R}$ follows.*

For locally confluent overlay TRSs[4], innermost termination and termination coincide [55]. Since terminating TRSs are μ-terminating, we have the following.

Corollary 2. *A locally confluent overlay TRS $\mathcal{R}$ is (innermost) terminating if and only if it is $\mu^{\mathsf{UA}}_{\mathcal{R}}$-terminating.*

Although the experiments in [3] suggest that, for the purpose of proving innermost termination of TRSs, Theorem 2 is weaker than the use of direct methods like the ones reported in [11, 51], Fernández's work inspired successful approaches for the use of *CSR* in complexity analysis (see Section 7 below).

4.3. Termination of outermost rewriting

In *outermost* rewriting (written $s \to_{\mathsf{o}} t$), reduction steps are performed at outermost redexes, i.e., those which are not contained in any other redex. In [32, 33] a transformation Δ^π from TRSs $\mathcal{R}$ to CS-TRSs $(\Delta^\pi\mathcal{R}, \mu)$ so that μ-termination of $\Delta^\pi\mathcal{R}$ implies outermost termination of $\mathcal{R}$ is defined. Here, Δ^π *marks* possible outermost redex positions by using an appropriate (semantic) *labelling* π.[5] Then, μ disallows rewritings in the arguments of marked symbols. In this way, outermost sequences with $\mathcal{R}$ are simulated as μ-rewriting sequences with $\Delta^\pi\mathcal{R}$.

Example 6. *For $\mathcal{R}$ in [33, Example 5.9]*

$$\mathsf{g}(x,x) \to \mathsf{f}(\mathsf{f}(x,x),x) \qquad \mathsf{f}(x,x) \to \mathsf{g}(x,x) \qquad \mathsf{f}(x,y) \to y$$

f *and* g *are marked as* $\mathsf{f}^{\perp,\perp}$ *and* $\mathsf{g}^{\perp,\perp}$*;* $\mu(\mathsf{f}^{\perp,\perp}) = \varnothing$ *and* $\mu(\mathsf{g}^{\perp,\perp}) = \{1,2\}$*. Here, μ differs for* $\mathsf{f}^{\perp,\perp}$ *and* $\mathsf{g}^{\perp,\perp}$ *due to the left-linear rule for* f*, which guarantees that* $\mathsf{f}(t,t')$ *is an outermost redex for all terms t and t'. In an outermost computation $s_1 \to_{\mathsf{o}} s_2 \to_{\mathsf{o}} \cdots \to_{\mathsf{o}} s_n$ with $\mathcal{R}$, for all $i \geq 1$ either $s_i = \mathsf{f}(t_i,t'_i)$ or $s_i = \mathsf{g}(t_i,t'_i)$ for some terms t_i, t'_i (except, perhaps, for s_n, which could be a variable). In the first case, s_i is an outermost redex and no reduction is required on t_i or t'_i. Hence, we can safely let $\mu(\mathsf{f}^{\perp,\perp}) = \varnothing$. In the second case, it is unclear whether s_i is a redex or not, hence we may need to explore t_i and t'_i to find the outermost redex. Hence, we have to let $\mu(\mathsf{g}^{\perp,\perp}) = \{1,2\}$. Finally, $\Delta^\pi\mathcal{R}$ is*

$$\mathsf{g}^{\perp,\perp}(x,x) \to \mathsf{f}^{\perp,\perp}(\mathsf{f}^{\perp,\perp}(x,x),x) \qquad \mathsf{f}^{\perp,\perp}(x,x) \to \mathsf{g}^{\perp,\perp}(x,x) \qquad \mathsf{f}^{\perp,\perp}(x,y) \to y$$

Thus, there is a μ-rewriting sequence $s_1^{\perp,\perp} \hookrightarrow_\mu s_2^{\perp,\perp} \hookrightarrow_\mu \cdots \hookrightarrow_\mu s_n^{\perp,\perp}$ with $\Delta^\pi\mathcal{R}$ for the marked versions $s_i^{\perp,\perp}$ of terms s_i (where symbols f are replaced by $f^{\perp,\perp}$) reducing the same redexes as the original outermost sequence.

---

[4] i.e., a TRS whose critical pairs $\langle \sigma(\ell)[\sigma(r')]_p, \sigma(r)\rangle = \langle s,t\rangle$ are *overlays* (i.e., $p = \Lambda$) and *convergent*, i.e., there is a term u such that $s \to^*_{\mathcal{R}} u$ and $t \to^*_{\mathcal{R}} u$.

[5] In semantic labelling function symbols f in terms are marked using labels λ from a given set of labels under the guidance of an algebraic interpretation $\mathcal{A}$ of the function symbols [130].

In the following results we omit the exact conditions for the labelling π, as we do not provide sufficient background here; full details can be found in [33, Theorem 5.8].

Theorem 3. [33, Theorem 5.8] *If $\Delta^\pi\mathcal{R}$ is μ-terminating, then $\mathcal{R}$ is outermost ground terminating.*

$\Delta^\pi\mathcal{R}$ in Example 6 is proved μ-terminating with MU-TERM. This proves $\mathcal{R}$ outermost terminating. For (quasi-)left-linear[6] TRSs (including left-linear TRSs), the transformation *characterizes* outermost ground termination as termination of *CSR*.

Theorem 4. [33, Theorem 5.13] *If $\mathcal{R}$ is quasi-left-linear and outermost ground terminating, then $\Delta^\pi\mathcal{R}$ is μ-terminating.*

5. *CSR* in the analysis of conditional rewriting

Recall from [116, Section 7] the usual notions and notations regarding Conditional Term Rewriting Systems (CTRSs). Conditional rules are written $\ell \to r \Leftarrow c$, where ℓ and r are respectively called the left- and right-hand side of the rule, and the *conditional part* c is a sequence $s_1 \approx t_1, \cdots, s_n \approx t_n$, where s_i, t_i are terms. Terms s and t of a condition $s \approx t$ are called the *left*- and *right*- hand side of the condition, respectively (*lhs* and *rhs* for short). A CTRS $\mathcal{R}$ whose rules satisfy $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(\ell) \cup \mathcal{V}ar(c)$ is called *deterministic* (DCTRS) if they all satisfy $\mathcal{V}ar(s_i) \subseteq \mathcal{V}ar(\ell) \cup \bigcup_{j=1}^{i-1} \mathcal{V}ar(t_j)$ for all $1 \leq i \leq n$.

When dealing with *oriented* CTRSs, conditions $s_i \approx t_i$ for $1 \leq i \leq n$ are treated as reachability tests $\sigma(s_i) \to^* \sigma(t_i)$ from (instances of) s_i to (instances of) t_i.[7] We consider the generic inference system

$$\mathfrak{I}_{\mathrm{CTRS}}[\mathbb{S}, \mathbb{R}] = \{(\mathrm{Rf}), (\mathrm{T})\} \cup \{(\mathrm{C})_{f,i} \mid f \in \mathbb{S}, 1 \leq i \leq ar(f)\} \cup \{(\mathrm{CRl})_\alpha \mid \alpha \in \mathbb{R}\}$$

where (Rf), (T), and $(\mathrm{C})_{f,i}$ are as in Figure 2, and $(\mathrm{CRl})_\alpha$ is

$$(\mathrm{CRl})_\alpha \quad \frac{s_1 \to^* t_1 \quad \cdots \quad s_n \to^* t_n}{\ell \to r}$$

for a rule $\alpha : \ell \to r \Leftarrow s_1 \approx t_1, \ldots, s_n \approx t_n$. Given an oriented CTRS $\mathcal{R} = (\mathcal{F}, R)$, an inference system $\mathcal{I}(\mathcal{R}) = \mathfrak{I}_{\mathrm{CTRS}}[\mathcal{F}, \mathcal{R}]$ is obtained. We write $s \to_{\mathcal{R}} t$ (resp. $s \to^*_{\mathcal{R}} t$) iff there is a proof tree for $s \to t$ (resp. $s \to^* t$) using $\mathcal{I}(\mathcal{R})$.

[6] A TRS $\mathcal{R}$ is *quasi-left-linear* if for all $\ell \to r \in \mathcal{R}$, ℓ is an instance of a linear *lhs* ℓ' for some $\ell' \to r' \in \mathcal{R}$ [120].

[7] Alternative treatments can be given, see [116, Definition 7.1.3].

5.1. Proving operational termination of CTRSs

Operational termination of a CTRS $\mathcal{R}$ is defined as the absence of infinite proof trees for goals $s \to^* t$ in $\mathcal{I}(\mathcal{R})$ [87], see also [90, Section 3]. Early attempts to prove operational termination of CTRSs involved

1. the use of *well-founded orderings* to compare the different components of conditional rules [74, 72, 21] (in particular, *quasi-decreasingness* [116, Definition 7.2.39], which is equivalent to operational termination [87]; we silently use this equivalence in the following), and
2. the development of *transformation* techniques (starting from Marchiori's *unravelings* [93]) to prove operational termination of CTRSs as termination of TRSs, see also [46, 115].

In this second approach, the following transformation $\mathcal{U}$ for oriented DCTRSs has been widely used, see [115, Definition 5] and also [46, page 45]. Each conditional rule $\alpha : \ell \to r \Leftarrow s_1 \approx t_1, \ldots, s_n \approx t_n$ is transformed into $n+1$ unconditional rules [116, Definition 7.2.48]:

$$\begin{array}{rcll} \ell & \to & U_1^\alpha(s_1, \vec{x}_1) & \\ U_{i-1}^\alpha(t_{i-1}, \vec{x}_{i-1}) & \to & U_i^\alpha(s_i, \vec{x}_i) & \quad 2 \leq i \leq n \\ U_n^\alpha(t_n, \vec{x}_n) & \to & r & \end{array}$$

where U_i^α are fresh new symbols and $\vec{x}_i$ are vectors of variables containing (for a given ordering on variables) the ordered sequence of the variables in $\mathcal{V}ar(\ell) \cup \mathcal{V}ar(t_1) \cup \cdots \cup \mathcal{V}ar(t_{i-1})$ for $1 \leq i \leq n$.

Example 7. *For the CTRS $\mathcal{R}$ in [46, p. 46] (left) we show $\mathcal{U}(\mathcal{R})$ (right):*

$$\begin{array}{rcll} \mathsf{a} & \to & \mathsf{b} & \\ \mathsf{f}(\mathsf{a}) & \to & \mathsf{b} & \\ \mathsf{g}(x) & \to & \mathsf{g}(\mathsf{a}) \Leftarrow \mathsf{f}(x) \approx x & (2) \end{array} \qquad \begin{array}{rcl} \mathsf{a} & \to & \mathsf{b} \\ \mathsf{f}(\mathsf{a}) & \to & \mathsf{b} \\ \mathsf{g}(x) & \to & U(\mathsf{f}(x), x) \\ U(x, x) & \to & \mathsf{g}(\mathsf{a}) \end{array}$$

The transformation is sound for proving operational termination, i.e., if $\mathcal{U}(\mathcal{R})$ is terminating, then $\mathcal{R}$ is operationally terminating [116, Proposition 7.2.50]. However, it is *not* complete, as there are operationally terminating TRSs $\mathcal{R}$ such that $\mathcal{U}(\mathcal{R})$ is *not* terminating.

Example 8. *As noticed by Giesl and Arts, although $\mathcal{R}$ in Example 7 is operationally terminating,*[8] *$\mathcal{U}(\mathcal{R})$ is* not *terminating:*

$$\underline{\mathsf{g}(\mathsf{a})} \to_{\mathcal{U}(\mathcal{R})} U(\underline{\mathsf{f}(\mathsf{a})}, \mathsf{a}) \to_{\mathcal{U}(\mathcal{R})} U(\mathsf{b}, \underline{\mathsf{a}}) \to_{\mathcal{U}(\mathcal{R})} \underline{U(\mathsf{b}, \mathsf{b})} \to_{\mathcal{U}(\mathcal{R})} \underline{\mathsf{g}(\mathsf{a})} \to_{\mathcal{U}(\mathcal{R})} \cdots \quad (3)$$

In the following, we discuss how *CSR* has been used to improve the use of orderings and transformations in proofs of operational termination.

[8] Actually, Giesl and Arts proved $\mathcal{R}$ *quasi-reductive*, which implies *quasi-decreasingness* of $\mathcal{R}$ (see [116, Section 7] for the definitions of these concepts and results relating them) and hence operational termination [87].

5.1.1. Context-sensitive quasi-reductivity

Let $\rhd_\mu$ be the strict *active subterm* relation, i.e., $s \rhd_\mu t$ iff $t = s|_p$ for some $p \in \mathcal{P}os^\mu(s) - \{\Lambda\}$. A DCTRS $\mathcal{R} = (\mathcal{F}, R)$ is *context-sensitively (cs-)quasi-reductive* if there is an extension $\mathcal{F}'$ of $\mathcal{F}$ ($\mathcal{F} \subseteq \mathcal{F}'$), a replacement map μ such that $\mu(f) = \{1, \ldots, ar(f)\}$ for all $f \in \mathcal{F}$, and a μ-monotonic, well-founded partial order $\succ_\mu$ on $\mathcal{T}(\mathcal{F}', \mathcal{X})$ such that, for every rule $\ell \to r \Leftarrow s_1 \approx t_1, \ldots, s_n \approx t_n$, every substitution σ and every i, $0 \leq i < n$, (1) if $\sigma(s_j) \succeq_\mu \sigma(t_j)$ for every $1 \leq j \leq i$, then $\sigma(\ell) \succ_\mu^{st} \sigma(s_{i+1})$, where $\succ_\mu^{st}$ is the transitive closure of $\succ_\mu \cup \rhd_\mu$, and (2) if $\sigma(s_j) \succeq_\mu \sigma(t_j)$ for every $1 \leq j \leq n$, then $\sigma(\ell) \succ_\mu \sigma(t)$ [123]. Schernhammer and Gramlich prove that cs-quasi-reductivity suffices for operational termination of DCTRSs.

Theorem 5. [123, Corollary 1] *Every cs-quasi-reductive DCTRS $\mathcal{R}$ is operationally terminating.*

Schernhammer and Gramlich do not try to use cs-quasi-reductivity as a direct technique for proving operational termination. Instead, they show that cs-quasi-reductivity is implied by the termination of the TRSs obtained by using a refined version of transformation $\mathcal{U}$. Then, such a transformation is used in practice. We discuss this in the following section.

5.1.2. Improving transformation $\mathcal{U}$

In [26, Section 3.2] an *optimized* version $\mathcal{U}_{opt}$[9] of $\mathcal{U}$ was introduced where the number of variables $\vec{x}_i$ stored in the right-hand sides $U_i^\alpha(s_i, \vec{x}_i)$ of the rules was reduced to avoid keeping track of unused variables as follows:

$$\begin{aligned}\vec{y}_i = {} & (\mathcal{V}ar(\ell) \cup \mathcal{V}ar(t_1) \cup \cdots \cup \mathcal{V}ar(t_{i-1})) \\ & \cap (\mathcal{V}ar(t_i) \cup \mathcal{V}ar(s_{i+1}) \cup \mathcal{V}ar(t_{i+1}) \cup \cdots \cup \mathcal{V}ar(s_n) \cup \mathcal{V}ar(t_n) \cup \mathcal{V}ar(r))\end{aligned}$$

Note that, for each sequence $\vec{x}_i$ in a rule of $\mathcal{U}(\mathcal{R})$ now we have a (possibly) shorter sequence $\vec{y}_i$ of variables. The following example shows the difference between $\mathcal{U}$ and $\mathcal{U}_{opt}$.

Example 9. *Consider the following CTRS $\mathcal{R}$ [123, Example 16]:*

$$\begin{array}{rcl}\mathsf{f}(x) & \to & \mathsf{c} \Leftarrow \mathsf{a} \to^* \mathsf{b} \\ \mathsf{g}(x,x) & \to & \mathsf{g}(\mathsf{f}(\mathsf{a}), \mathsf{f}(\mathsf{b}))\end{array}$$

We obtain the following TRSs $\mathcal{U}(\mathcal{R})$ (left) and $\mathcal{U}_{opt}(\mathcal{R})$ (right):

$$\begin{array}{rclcrcl}\mathsf{f}(x) & \to & U(\mathsf{a}, x) & \qquad & \mathsf{f}(x) & \to & U(\mathsf{a}) \\ U(\mathsf{b}, x) & \to & \mathsf{c} & & U(\mathsf{b}) & \to & \mathsf{c} \\ \mathsf{g}(x,x) & \to & \mathsf{g}(\mathsf{f}(\mathsf{a}), \mathsf{f}(\mathsf{b})) & & \mathsf{g}(x,x) & \to & \mathsf{g}(\mathsf{f}(\mathsf{a}), \mathsf{f}(\mathsf{b}))\end{array}$$

For $\mathcal{R}$ in Example 7, though, there is no difference.

[9]This notation is taken from [123, Section 7].

Example 10. *For $\mathcal{R}$ in Example 7, and the conditional rule (2), we have $\vec{x}_1 = \mathcal{V}ar(\mathsf{g}(x)) \cap (\mathcal{V}ar(x) \cup \mathcal{V}ar(\mathsf{g}(\mathsf{a})) = \{x\}$, i.e., $\mathcal{U}(\mathcal{R})$ and $\mathcal{U}_{opt}(\mathcal{R})$ coincide.*

In [26] an additional refinement was proposed to avoid infinite sequences like (3) thus extending the use of the transformation in proofs of operational termination. The idea is to use a replacement map $\mu^{\mathcal{U}}$ to restrict reductions on the variable-storage part of the new symbols U_i^α as follows: for all (oriented, conditional) rules $\alpha : \ell \to r \Leftarrow s_1 \approx t_1, \cdots, s_n \approx t_n$ and $1 \leq i \leq n$,

$$\mu^{\mathcal{U}}(U_i^\alpha) = \{1\}$$

and $\mu^{\mathcal{U}}(f) = \mu_\top(f)$ for any other symbol f.

Example 11. *For $\mathcal{R}$ in Example 7, we have $\mu^{\mathcal{U}}(\mathsf{f}) = \mu^{\mathcal{U}}(\mathsf{g}) = \mu^{\mathcal{U}}(U) = \{1\}$.*

If $\mathcal{U}_{opt}(\mathcal{R})$ is $\mu^{\mathcal{U}}$-terminating, then $\mathcal{R}$ is operationally terminating [23, Theorem 2], i.e., the new transformation is *sound* for proving operational termination of CTRSs. Unfortunately, it remains *incomplete*.

Example 12. *Consider $\mathcal{R}$, $\mathcal{U}(\mathcal{R})$, and $\mathcal{U}_{opt}(\mathcal{R})$ as in Example 9. $\mathcal{U}(\mathcal{R})$ is terminating (and hence $\mathcal{R}$ is operationally terminating). However, $\mathcal{U}_{opt}(\mathcal{R})$ is* not *$\mu^{\mathcal{U}}$-terminating [123, Example 16]:*

$$\mathsf{g}(\mathsf{f}(\mathsf{a}), \mathsf{f}(\mathsf{b})) \hookrightarrow^+_{\mathcal{U}_{opt}(\mathcal{R}),\mu^{\mathcal{U}}} \mathsf{g}(U(\mathsf{a}), U(\mathsf{a})) \hookrightarrow_{\mathcal{U}_{opt}(\mathcal{R}),\mu^{\mathcal{U}}} \mathsf{g}(\mathsf{f}(\mathsf{a}), \mathsf{f}(\mathsf{b})) \hookrightarrow_{\mathcal{U}_{opt}(\mathcal{R}),\mu^{\mathcal{U}}} \cdots \quad (4)$$

Schernhammer and Gramlich proved that $\mu^{\mathcal{U}}$-termination of $\mathcal{U}(\mathcal{R})$ implies cs-quasi-reductivity of $\mathcal{R}$ [123, Theorem 3]; hence, by Theorem 5, operational termination of $\mathcal{R}$ follows. The last sentence in [123, footnote 17] says that *$\mu^{\mathcal{U}}$-termination of $\mathcal{U}_{opt}(\mathcal{R})$ implies $\mu^{\mathcal{U}}$-termination of $\mathcal{U}(\mathcal{R})$*. No formal proof is given, though. In this setting, the following question naturally arises: is proving $\mu^{\mathcal{U}}$-termination of $\mathcal{U}(\mathcal{R})$ strictly better (for the purpose of proving operational termination of CTRSs $\mathcal{R}$) than just proving termination of $\mathcal{U}(\mathcal{R})$, as usually done in termination tools like AProVE [47]? We can give a *positive* answer.

Proposition 1.$^{(\star)}$ *There is a CTRS $\mathcal{R}$ which can be proved operationally terminating as the $\mu^{\mathcal{U}}$-termination of $\mathcal{U}(\mathcal{R})$ whereas $\mathcal{U}(\mathcal{R})$ is* not *terminating.*

Proof. Appendix A proves that, for $\mathcal{R}$ in Example 7 and $\mu^{\mathcal{U}}$ in Example 11, $\mathcal{U}(\mathcal{R})$ (which coincides with $\mathcal{U}_{opt}(\mathcal{R})$, see Example 10) is $\mu^{\mathcal{U}}$-terminating. Recall that $\mathcal{U}(\mathcal{R})$ is *not* terminating (see (3)). Thus, this proves the desired fact. □

Since termination of $\mathcal{U}(\mathcal{R})$ implies the $\mu^{\mathcal{U}}$-termination of $\mathcal{U}(\mathcal{R})$, Proposition 1 shows that, for the purpose of proving operational termination of CTRSs $\mathcal{R}$, proving $\mu^{\mathcal{U}}$-termination of $\mathcal{U}(\mathcal{R})$ is *more powerful* than just proving termination of $\mathcal{U}(\mathcal{R})$. Finally, a main contribution of [123] was the following *completeness* result restricted to *terms of the original signature $\mathcal{F}$*.

Theorem 6. [123, Theorem 4] *Let $\mathcal{R} = (\mathcal{F}, R)$ be a DCTRS. If $\mathcal{R}$ is operationally terminating, then $\mathcal{U}(\mathcal{R})$ is $\mu^{\mathcal{U}}$-terminating on $\mathcal{T}(\mathcal{F}, \mathcal{X})$.*

5.2. Soundness and completeness of unravelings for CTRSs

The use of transformations for *implementing* rewriting with CTRSs $\mathcal{R}$ using TRSs has been investigated by several authors. Interesting summaries with many pointers to the literature can be found in [124, Section 2] and [112, Section 1]. Nishida, Sakabe and Sakai investigated the use of Marchiori's unraveling transformations. In [110, 111], they show that transformation $\mathcal{U}$ above

> "is sound for a 3-DCTRS[10] $\mathcal{R}$ if the reduction of $\mathcal{U}(\mathcal{R})$ is restricted to context-sensitive rewriting with the replacement map μ such that $\mu(U_i) = \{1\}$ $(\cdots)$ the replacement map forbids reducing any redex inside the second or later arguments of U symbols." [112, p. 9]

Note that the aforementioned replacement map μ is just $\mu^{\mathcal{U}}$ above. This means that for all terms $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, if $s \hookrightarrow^*_{\mathcal{U}(\mathcal{R}),\mu^{\mathcal{U}}} t$, then $s \to^*_{\mathcal{R}} t$ (*soundness*). Replacement restrictions play an important role in guaranteeing this result.

Example 13. *For $\mathcal{R}$ and $\mathcal{U}(\mathcal{R})$ in Example 7, the sequence (3) shows that* $\mathsf{g}(\mathsf{a}) \to^+_{\mathcal{U}(\mathcal{R})} \mathsf{g}(\mathsf{a})$*. However, since $\mathcal{R}$ is operationally terminating,* $\mathsf{g}(\mathsf{a}) \to^+_{\mathcal{R}} \mathsf{g}(\mathsf{a})$ *does not hold, i.e., $\mathcal{U}$ is* not *sound. As observed in [23, Example 5], this problem is avoided by* CSR *using $\mu^{\mathcal{U}}$, which forbids the third step of the sequence.*

Note that [23] did *not* investigate how to achieve soundness of $\mathcal{U}$ by using *CSR*. The focus of [26, 23] was improving $\mathcal{U}$ into $\mathcal{U}_{opt}$ for proving operational termination of CTRSs. Indeed, $\mathcal{U}_{opt}$ is *not* sound either.

Example 14. *For $\mathcal{R}$, $\mathcal{U}_{opt}$, and $\mu^{\mathcal{U}}$ in Example 12, the first part of (4) shows that* $\mathsf{g}(\mathsf{f}(\mathsf{a}), \mathsf{f}(\mathsf{b})) \hookrightarrow^+_{\mathcal{U}_{opt}(\mathcal{R}),\mu^{\mathcal{U}}} \mathsf{g}(\mathsf{f}(\mathsf{a}), \mathsf{f}(\mathsf{b}))$*. However, since $\mathcal{R}$ is operationally terminating,* $\mathsf{g}(\mathsf{f}(\mathsf{a}), \mathsf{f}(\mathsf{b})) \to^+_{\mathcal{R}} \mathsf{g}(\mathsf{f}(\mathsf{a}), \mathsf{f}(\mathsf{b}))$ *does not hold, i.e., $\mathcal{U}_{opt}$ is* not *sound.*

Other variants of $\mathcal{U}$ have been considered and replacement restrictions successfully used to guarantee soundness and completeness [110, 108]. Schernhammer and Gramlich proved $\mathcal{U}$ *complete*, i.e., for all terms $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, $s \to_{\mathcal{R}} t$ implies $s \hookrightarrow^+_{\mathcal{U}(\mathcal{R}),\mu^{\mathcal{U}}} t$ (this was proved for $\mathcal{U}_{opt}$ in [23, Lemma 3]) and also provided a soundness result which does not require the *membership condition* in [110].

Theorem 7. *Let $\mathcal{R}$ be a DCTRS and $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$.*

- *(Completeness [123, Theorem 1]) If $s \to_{\mathcal{R}} t$, then $s \hookrightarrow^+_{\mathcal{U}(\mathcal{R}),\mu^{\mathcal{U}}} t$.*
- *(Soundness [123, Theorem 2]) If $s \hookrightarrow^+_{\mathcal{U}(\mathcal{R}),\mu^{\mathcal{U}}} t$, then $s \to^+_{\mathcal{R}} t$.*

Remark 2 (Soundness of $\mathcal{U}$ and use of $\mu^{\mathcal{U}}$). *In contrast to completeness, which holds for any replacement map μ less restrictive than $\mu^{\mathcal{U}}$, i.e., $\mu^{\mathcal{U}} \sqsubseteq \mu$, it is, in general,* false *that for all $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, $s \hookrightarrow^+_{\mathcal{U}(\mathcal{R}),\mu} t$ implies $s \to^+_{\mathcal{R}} t$. For instance, since $\to_{\mathcal{U}(\mathcal{R})} = \hookrightarrow_{\mathcal{U}(\mathcal{R}),\mu_\top}$, sequence (3) provides a counterexample.*

[10] That is, a DCTRS whose rules $\ell \to r \Leftarrow s_1 \approx t_1, \ldots, s_n \approx t_n$ are 3-rules [116, Definition 7.1.1], i.e., $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(\ell) \cup \bigcup_{i=1}^n \mathcal{V}ar(s_i) \cup \mathcal{V}ar(t_i)$ holds for all of them.

The following result can be used to (dis)prove confluence of DCTRSs by using $\mathcal{U}$ and $\mathcal{U}_{opt}$ together with $\mu^{\mathcal{U}}$. Note that we require *termination* rather than *operational termination* of CTRSs (which of course implies the former, but not vice versa, see [90]). A CTRS is terminating if $\to_{\mathcal{R}}$ is terminating.

Theorem 8.$^{(\star)}$ *Let $\mathcal{R}$ be a DCTRS.*

1. *If $\mathcal{R}$ is terminating and $\mathcal{U}_{opt}(\mathcal{R})$ (or $\mathcal{U}(\mathcal{R})$) is confluent, then $\mathcal{R}$ is confluent.*
2. *If $\mathcal{U}(\mathcal{R})$ has a $\mu^{\mathcal{U}}$-critical pair $\langle s,t\rangle$ such that (i) $s \in \mathcal{T}(\mathcal{F},\mathcal{X})$, (ii) $t \notin \mathcal{T}(\mathcal{F},\mathcal{X})$, (iii) $t \hookrightarrow^*_{\mathcal{U}(\mathcal{R}),\mu^{\mathcal{U}}} t'$ for some $t' \in \mathcal{T}(\mathcal{F},\mathcal{X})$, and (iv) $\langle s,t\rangle$ is not $\hookrightarrow^*_{\mathcal{U}(\mathcal{R}),\mu^{\mathcal{U}}}$-joinable, then $\mathcal{R}$ is not confluent.*

Proof. In both cases, we proceed by contradiction.

1. If $\mathcal{R}$ is not confluent, then there are terms $s,t,t' \in \mathcal{T}(\mathcal{F},\mathcal{X})$ such that $s \to^*_{\mathcal{R}} t$ and $s \to^*_{\mathcal{R}} t'$ but t and t' are not $\to^*_{\mathcal{R}}$-joinable. By termination of $\to_{\mathcal{R}}$, we can assume that t and t' are different irreducible terms[11] $t \neq t'$. By [23, Lemma 3] (or Theorem 7(1) for $\mathcal{U}$), $s \hookrightarrow^*_{\mathcal{U}_{opt}(\mathcal{R}),\mu^{\mathcal{U}}} t$ and $s \hookrightarrow^*_{\mathcal{U}_{opt}(\mathcal{R}),\mu^{\mathcal{U}}} t'$. Since $\mu^{\mathcal{U}} \sqsubseteq \mu_\top$, and $\hookrightarrow^*_{\mathcal{U}_{opt}(\mathcal{R}),\mu_\top} = \to^*_{\mathcal{U}_{opt}(\mathcal{R})}$ (and similarly for $U(\mathcal{R})$), we have $s \to^*_{\mathcal{U}_{opt}(\mathcal{R})} t$ and $s \to^*_{\mathcal{U}_{opt}(\mathcal{R})} t'$. By confluence of $\mathcal{U}_{opt}(\mathcal{R})$, there is u such that $t \to^*_{\mathcal{U}_{opt}(\mathcal{R})} u$ and $t' \to^*_{\mathcal{U}_{opt}(\mathcal{R})} u$. However, since t and t' are normal forms built from variables and symbols in $\mathcal{F}$ only, they are $\to_{\mathcal{U}_{opt}(\mathcal{R})}$-normal forms. Thus, $t = u = t'$, a contradiction.
2. If $\mathcal{R}$ is confluent but there is a $\mu^{\mathcal{U}}$-critical pair $\langle s,t\rangle$ satisfying (i)–(iv), then there is a conditional rule $\ell \to r \Leftarrow s_1 \approx t_1,\ldots,s_n \approx t_n \in \mathcal{R}$ and a rule $\ell' \to r' \in \mathcal{R}$ and $p \in \mathcal{P}os^{\mu^{\mathcal{U}}}_{\mathcal{F}}(\ell)$ such that $s = \sigma(\ell|_p)[\sigma(r')]_p$ and $t = U^\alpha_1(s_1,\vec{x})$ for the most general unifier σ of $\ell|_p$ and ℓ'. Note that $\sigma(\ell) \hookrightarrow_{\mathcal{U}(\mathcal{R}),\mu^{\mathcal{U}}} s$ and $\sigma(\ell) \hookrightarrow_{\mathcal{U}(\mathcal{R}),\mu^{\mathcal{U}}} t \hookrightarrow^*_{\mathcal{U}(\mathcal{R}),\mu^{\mathcal{U}}} t'$. Since $\sigma(\ell), s, t' \in \mathcal{T}(\mathcal{F},\mathcal{X})$, by [123, Theorem 2] (which applies to $\mu^{\mathcal{U}}$ only, not necessarilty to less restrictive replacement maps μ), $\sigma(\ell) \to^*_{\mathcal{R}} s$ and $\sigma(\ell) \to^*_{\mathcal{R}} t'$. By confluence of $\mathcal{R}$, there is u such that $s \to^*_{\mathcal{R}} u$ and $t' \to^*_{\mathcal{R}} u$. By [123, Theorem 1], $s \hookrightarrow^*_{\mathcal{U}(\mathcal{R}),\mu} u$ and $t' \hookrightarrow^*_{\mathcal{U}(\mathcal{R}),\mu} u$. Thus, $\langle s,t\rangle$ is $\hookrightarrow^*_{\mathcal{U}(\mathcal{R}),\mu^{\mathcal{U}}}$-joinable, a contradiction.

□

Termination of CTRSs can be specifically proved by using the results and techniques in [90, 91, 92], or automatically by proving operational termination of $\mathcal{R}$ using AProVE or mu-term. The platform CoCoWeb [66], permits the use of several confluence tools for (C)TRSs. As for (iii) and (iv) in Theorem 8(2):

[11] In conditional rewriting, distinguishing between *irreducible terms* (admitting no one-step conditional reduction) and *normal forms* (irreducible terms raising no infinite proof tree) [89, Definitions 5 & 6] is important. Here, termination of $\mathcal{R}$ guarantees irreducibility of t and t'.

(iii) For the critical pair $\langle s,t\rangle$, if t is ground, then $t \hookrightarrow^*_{\mathcal{U}(\mathcal{R}),\mu^{\mathcal{U}}} t'$ for some $t' \in \mathcal{T}(\mathcal{F},\mathcal{X})$ can be automatically proved as the *feasibility* of

$$t \hookrightarrow^* x, \mathit{test}(x) \hookrightarrow^* \mathsf{tt} \tag{5}$$

with respect to $\mathcal{U}(\mathcal{R}) \cup \mathit{Test}(\mathcal{F})$[12], where $\mathit{Test}(\mathcal{F})$ is the CTRS with rules

$$\begin{array}{rcl} \mathit{test}(a) & \to & \mathsf{tt} \\ \mathit{test}(b(x)) & \to & \mathit{test}(x) \\ \mathit{test}(f(x_1,\ldots,x_k)) & \to & \mathsf{tt} \Leftarrow \mathit{test}(x_1) \approx \mathsf{tt},\ldots,\mathit{test}(x_k) \approx \mathsf{tt} \end{array}$$

for all constant symbols a, monadic symbols b and k-ary symbols f for $k > 1$ belonging to $\mathcal{F}$, and a new symbol *test*. Here, $\mu^{\mathcal{U}}$ can be extended to *test* by $\mu^{\mathcal{U}}(\mathit{test}) = \varnothing$, although $\mu^{\mathcal{U}}(\mathit{test}) = \{1\}$ is also valid.[13] If (5) is *feasible*, then a ground term t' exists that satisfies (iii). The tool infChecker[14] [59] is able to deal with such kind of (in)feasibility problems involving *CSR*.

(iv) The non-$\hookrightarrow^*_{\mathcal{U}(\mathcal{R}),\mu^{\mathcal{U}}}$-joinability of $\langle s,t\rangle$ can be proved as the *infeasibility* of

$$s \hookrightarrow^* x,\; t \hookrightarrow^* x \tag{6}$$

with respect to $\mathcal{U}(\mathcal{R})$ and $\mu^{\mathcal{U}}$, which, again, can be proved using infChecker. Infeasibility tools in CoCoWeb not supporting *CSR* can also be used to prove infeasibility of $s \to^* x, t \to^* x$, which implies infeasibility of (6).

Example 15. *Consider the following CTRS $\mathcal{R}$ (left) and $\mathcal{U}_{opt}(\mathcal{R})$ (right):*

$$\begin{array}{rcl} \mathsf{b} & \to & \mathsf{f}(\mathsf{a}) \\ \mathsf{g}(\mathsf{a}) & \to & \mathsf{c}(\mathsf{a}) \\ \mathsf{f}(x) & \to & y \Leftarrow \mathsf{g}(x) \approx \mathsf{c}(y), x \approx y \end{array} \qquad \begin{array}{rcl} \mathsf{b} & \to & \mathsf{f}(\mathsf{a}) \\ \mathsf{g}(\mathsf{a}) & \to & \mathsf{c}(\mathsf{a}) \\ \mathsf{f}(x) & \to & U_1(\mathsf{g}(x),x) \\ U_1(\mathsf{c}(y),x) & \to & U_2(x,y) \\ U_2(y,y) & \to & y \end{array}$$

Operational termination of $\mathcal{R}$ can be proved with MU-TERM. *All confluence tools in CoCoWeb proved $\mathcal{U}_{opt}(\mathcal{R})$ confluent, thus concluding confluence of $\mathcal{R}$.*

Although a direct proof of confluence of $\mathcal{R}$ in Example 15 can be obtained by using several tools in CoCoWeb, we failed to obtain a proof with CO3 [107] (also using the online version of CO3[15]). The fact that CO3 fails to prove confluence

[12] If there is a substitution σ such that $t \hookrightarrow^*_{\mathcal{U}(\mathcal{R}),\mu^{\mathcal{U}}} \sigma(x)$ and $\mathit{test}(\sigma(x)) \hookrightarrow^*_{\mathit{Test}(\mathcal{F}),\mu^{\mathcal{U}}} \mathsf{tt}$, then (5) is *feasible*; otherwise, it is *infeasible*.

[13] In the feasibility framework [59], (5) could be more precisely given as $t \hookrightarrow^* x, \mathit{test}(x) \to^* \mathsf{tt}$ with $\hookrightarrow^*$ defined by $\mathcal{U}(\mathcal{R})$ and $\to$ defined by $\mathit{Test}(\mathcal{F})$ without overlapping them. Unfortunately, this is not supported by any tool yet.

[14] `http://zenon.dsic.upv.es/infChecker/`

[15] `https://www.trs.cm.is.nagoya-u.ac.jp/co3/wui.php`

of $\mathcal{R}$ in Example 15 shows that Theorem 8 complements existing results on proving confluence of CTRSs by using transformations [93, 124, 113]. Actually, CO3 uses both $\mathcal{U}$ and Serbanuta and Roșu's transformation SR [124]. In their description of CO3 [107], the authors write:

> The main technique in this tool is based on the following theorem: a weakly left-linear normal 1-CTRS $\mathcal{R}$ is confluent if one of $\mathsf{SR}(\mathcal{R})$ and $\mathcal{U}(\mathcal{R})$ is confluent [113]

1-CTRSs consist of rules $\ell \to r \Leftarrow c$ satisfying $\mathcal{V}ar(r) \cup \mathcal{V}ar(c) \subseteq \mathcal{V}ar(\ell)$; and in *normal* CTRSs [116, Definition 7.1.3], the *rhs*'s t_i of conditions $s_i \to t_i$ are *ground* and contain no preredex.[16] Note that $\mathcal{R}$ in Example 15 is *neither* normal nor a 1-CTRS. Theorem 8 does not require CTRSs to be normal or 1-CTRSs (but termination is required). Theorem 8 is also useful to *disprove* confluence.

Example 16. *Consider the following CTRS $\mathcal{R}$ (left) and $\mathcal{U}(\mathcal{R})$ (right):*

$$\begin{array}{rcll} \mathsf{b} & \to & \mathsf{f}(\mathsf{a}) & (7) \\ \mathsf{g}(x) & \to & \mathsf{c}(x) & (8) \\ \mathsf{a} & \to & \mathsf{d} & (9) \\ \mathsf{f}(\mathsf{a}) & \to & y \Leftarrow \mathsf{d} \approx x, \mathsf{g}(x) \approx \mathsf{c}(y) & (10) \end{array} \qquad \begin{array}{rcll} \mathsf{b} & \to & \mathsf{f}(\mathsf{a}) & (11) \\ \mathsf{g}(x) & \to & \mathsf{c}(x) & (12) \\ \mathsf{a} & \to & \mathsf{d} & (13) \\ \mathsf{f}(\mathsf{a}) & \to & U_1(\mathsf{d}) & (14) \\ U_1(x) & \to & U_2(\mathsf{g}(x), x) & (15) \\ U_2(\mathsf{c}(y), x) & \to & y & (16) \end{array}$$

Note that $\mathcal{R}$ is not *confluent as we have:*

$$\mathsf{f}(\underline{\mathsf{a}}) \to_{(9)} \mathsf{f}(\mathsf{d}) \quad \textit{and} \quad \underline{\mathsf{f}(\mathsf{a}))} \to_{(10)} \mathsf{d}$$

In the last case, this is due to the use of substitution $\sigma(x) = \mathsf{d}$, $\sigma(y) = \mathsf{d}$ and because $\mathsf{d} = \sigma(x)$ and $\sigma(\mathsf{g}(x)) = \mathsf{g}(\mathsf{d}) \to_{(8)} \mathsf{c}(\mathsf{d}) = \sigma(\mathsf{c}(y))$. Both $\mathsf{f}(\mathsf{d})$ and d are normal forms. No tool in CoCoWeb, though, was able to disprove confluence of $\mathcal{R}$. The only critical pair of $\mathcal{U}(\mathcal{R})$ is $\langle \mathsf{f}(\mathsf{d}), U_1(\mathsf{d})\rangle$ and we have:

$$\underline{U_1(\mathsf{d})} \hookrightarrow_{(14)} U_2(\underline{\mathsf{g}(\mathsf{d})}, \mathsf{d}) \hookrightarrow_{(12)} \underline{U_2(\mathsf{c}(\mathsf{d}), \mathsf{d})} \hookrightarrow_{(16)} \mathsf{d} \in \mathcal{T}(\mathcal{F}, \mathcal{X})$$

Note that $\mathsf{f}(\mathsf{d}) \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ is a normal form and $U_1(\mathsf{d}) \notin \mathcal{T}(\mathcal{F}, \mathcal{X})$ is not reducible to $\mathsf{f}(\mathsf{d})$, i.e., the critical pair is not $\hookrightarrow^*_{\mathcal{U}(\mathcal{R}),\mu^{\mathcal{U}}}$-joinable. By Theorem 8, $\mathcal{R}$ is not confluent.

Soundness and completeness of $\mathcal{U}$ and μ with regard to *eager* computations using *innermost* reduction in CTRSs is also investigated in [109].

6. Productivity

Productivity in lazy functional languages "*captures the idea of computability, of progress of infinite-list programs. If an infinite-list program is productive, then every element of the list can be computed in finite 'time'*" [125].

[16] [13, Definition 4.1] uses *preredex* for instances $\sigma(\ell)$ of the left-hand sides ℓ of conditional rules $\ell \to r \Leftarrow c$. The term *redex* is reserved for *reducible* preredexes [13, Definition 2.4.1].

Remark 3 (Infinite terms). *Productivity often involves the possiblity of computing (i.e., approaching) infinite terms. Formally,* infinite terms *are partial functions* $t : \mathbb{N}^*_{>0} \to \mathcal{F} \cup \mathcal{X}$ *from sequences of positive numbers to symbols where the domain* $\mathcal{D}om(t)$ *is a* tree-domain*, i.e., an infinite set of positions satisfying (a)* $\mathcal{D}om(t)$ *is* prefix closed*, and (b) if* $p \in \mathcal{D}om(t)$ *and* $root(t) = f$*, then* $p.i \in \mathcal{D}om(t)$ *for all* $1 \leq i \leq ar(f)$ *[20, Section 1.2]. For instance, for* f *of arity* 1*,* $t = \mathsf{f}^\omega$ *is given by* $t(p) = \mathsf{f}$ *for all* $p \in 1^*$ *(i.e.,* $\mathcal{D}om(t) = 1^* = \{\Lambda, 1, 1.1, \ldots\}$*). Note, however, that, as usually done in the literature of productivity, we do* not *allow infinite terms in rules of TRSs.*

In term rewriting most presentations of productivity analysis use sorted signatures [53]. The set of sorts $\mathcal{S}$ is partitioned: $\mathcal{S} = \Delta \uplus \nabla$, where Δ is the set of *data sorts* (inductive datatypes like booleans, natural numbers, finite lists,...) and ∇ is the set of *codata* sorts (coinductive datatypes such as streams and infinite trees) [34, 132]. Terms of sort Δ (resp. ∇) are called *(co)data terms.* For a *ranked* symbol $f : \tau_1 \times \cdots \times \tau_n \to \tau$, denote as $ar_\Delta(f)$ (resp. $ar_\nabla(f)$) the number of arguments of f of sort Δ (resp. ∇). Data arguments come in the first $ar_\Delta(f)$ arguments of symbols. A *constructor* TRS $\mathcal{R}$ is a TRS with constructor symbols $\mathcal{C}$ where the left-hand sides ℓ of all rules $\ell \to r \in \mathcal{R}$ are of the form $f(\ell_1, \ldots, \ell_k)$ for constructor terms $\ell_1, \ldots, \ell_k \in \mathcal{T}(\mathcal{C}, \mathcal{X})$. A *tree specification* is a $(\Delta \uplus \nabla)$-sorted, orthogonal, exhaustive constructor TRS $\mathcal{R}$ [34, Definition 3.1]. Here, $\mathcal{R}$ is called *exhaustive* if for all $f \in \mathcal{F}$, every $f(t_1, \ldots, t_k)$ is a redex whenever t_i are (possiby infinite) ground constructor terms for $1 \leq i \leq k$.[17]

Remark 4 (*CSR* in sorted signatures). *Although our presentation of* CSR *pays no explicit attention to sorts, an extension of* CSR *to deal with ranked functions* $f : s_1 \cdots s_k \to s$ *and sorted Term Rerwriting Systems is immediate, see [63, 23]. The definition of canonical replacement map is also ported without changes. Rewriting sorted terms using* CSR *is done in the obvious way and then the notion of termination of* CSR *used below naturally arises.*

6.1. Termination of CSR *and constructor normalization*

A TRS $\mathcal{R}$ is *constructor normalizing* if every ground term $t \in \mathcal{T}(\mathcal{F})$ rewrites into a possibly infinite constructor normal form [34, Definition 3.5]. For left-linear TRSs $\mathcal{R}$ and canonical replacement maps $\mu \in CM_\mathcal{R}$, the μ-termination of $\mathcal{R}$ provides a sufficient condition for constructor normalization.

Theorem 9. [85, Theorem 4] *Let* $\mathcal{R}$ *be an exhaustive, left-linear TRS and* $\mu \in CM_\mathcal{R}$*. If* $\mathcal{R}$ *is* μ*-terminating, then* $\mathcal{R}$ *is constructor normalizing.*

Since tree specifications are left-linear and exhaustive, Theorem 9 holds for tree specifications as well.

[17] See the paragraph below [34, Definition 2.9] for a discussion regarding the relationship between exhaustiveness and the well-known property of *sufficient completeness* [75].

Example 17. *The following tree specification $\mathcal{R}$ (cf. [132, Example 4.6])*

$$\begin{array}{rcl}
\mathsf{p} & \to & \mathsf{zip}(\mathsf{alt}, \mathsf{p}) \\
\mathsf{alt} & \to & \mathsf{0} : \mathsf{1} : \mathsf{alt} \\
\mathsf{zip}(x : \sigma, \tau) & \to & x : \mathsf{zip}(\tau, \sigma)
\end{array}$$

(where no constant for empty lists is included!) is easily proved $\mu^{can}_{\mathcal{R}}$-terminating (use MU-TERM*). Note that $\mathcal{R}$ is* exhaustive *due to the sort discipline (for instance,* $\mathsf{zip}(\mathsf{0}, \mathsf{0})$ *is not allowed) and to the fact that no constructor for empty lists is provided (i.e., there is no finite list and all constructor lists are of the form $s : t$ for constructor terms s and t, where t is always infinite). By Theorem 9, $\mathcal{R}$ is constructor normalizing.*

With some additional conditions, constructor normalization is *characterized* by canonical termination of *CSR*. We say that a TRS $\mathcal{R}$ is *strongly compatible* iff $\mathcal{P}os^{\mu^{can}_{\mathcal{R}}}(\ell) = \mathcal{P}os_{\mathcal{F}}(\ell)$ for all $\ell \to r \in \mathcal{R}$ [85, Section 3.2].

Theorem 10. [85, Theorem 6] *Let $\mathcal{R} = (\mathcal{C} \uplus \mathcal{D}, R)$ be an orthogonal, strongly compatible TRS such that either (i) $\mu^{can}_{\mathcal{R}}(c) = \varnothing$ for all $c \in \mathcal{C}$, or (ii) $\mathcal{R}$ contains no rule $\ell \to x$ for some $x \in \mathcal{X}$ and $\mu^{can}_{\mathcal{R}}(c) = \varnothing$ for all $c \in \mathcal{C}$ such that $c = root(r)$ for some $\ell \to r \in \mathcal{R}$. If $\mathcal{R}$ is constructor normalizing, then it is $\mu^{can}_{\mathcal{R}}$-terminating.*

As remarked in [34, Section 3.2], several authors call $\mathcal{R}$ productive if it is constructor normalizing [30, 29, 31, 131, 132]. Zantema and Raffelsieper were the first to prove constructor normalization as termination of *CSR*.

Theorem 11. [132, Theorem 4.1] *Let $\mathcal{R} = (\mathcal{C} \uplus \mathcal{D}, R)$ be a* proper *tree specification and μ given by $\mu(f) = \{1, \ldots, ar(f)\}$ if $f \in \mathcal{D}$ and $\mu(c) = \{1, \ldots, ar_{\Delta}(c)\}$ if $c \in \mathcal{C}$. If $\mathcal{R}$ is μ-terminating, then $\mathcal{R}$ is constructor normalizing.*

Theorem 11 is a particular case of Theorem 9 because proper tree specifications are TRSs whose rules $\ell \to r$ have left-hand sides $\ell = f(t_1, \ldots, t_k)$, where t_i is either a variable or a *flat* constructor term $c_i(x_1, \ldots, x_m)$ for some constructor symbol c_i and variables $x_1, \ldots, x_m$. In this case, μ in Theorem 11 must be *canonical*, i.e., $\mu \in CM_{\mathcal{R}}$ [85].

6.2. Termination and productivity

Endrullis and Hendriks give a more elaborate (and restrictive) definition of productivity. Given a (possibly infinite) term t and $\Gamma \subseteq \mathcal{F}$, a Γ-*path* in t is a (finite or infinite) sequence $\langle p_1, c_1\rangle$, $\langle p_2, c_2\rangle, \ldots$ such that $c_i = root(t|_{p_i}) \in \Gamma$ and $p_{i+1} = p_i.j$ with $1 \leq j \leq ar(c_i)$ [34, Definition 3.7]. A tree specification is *data-finite* if for all ground terms $s \in \mathcal{T}(\mathcal{F})$ and (possibly infinite) constructor normal forms t of s, every $\mathcal{C}_{\Delta}$-path in t (containing data constructors only) is finite [34, Definition 3.8].

Definition 3. [34, Definition 3.11] *A tree specification $\mathcal{R}$ is* productive *if $\mathcal{R}$ is constructor normalizing and data-finite.*

In the following result, μ_Δ is given by $\mu_\Delta(c) = \{1,\ldots,ar_\Delta(c)\}$ for all $c \in \mathcal{C}_\Delta$, and $\mu_\Delta(f) = \varnothing$ for all other symbols f.

Theorem 12. [85, Theorem 5] *Let $\mathcal{R}$ be a left-linear, exhaustive TRS and $\mu \in CM_\mathcal{R}$ be such that $\mu_\Delta \sqsubseteq \mu$. If $\mathcal{R}$ is μ-terminating, then $\mathcal{R}$ is productive.*

Example 18. *For $\mathcal{R}$ in [34, Example 6.8], i.e.,*

$$\begin{array}{rcl@{\qquad}rcl}
x+\mathsf{0} & \to & x & x\times\mathsf{0} & \to & \mathsf{0}\\
x+\mathsf{S}(y) & \to & \mathsf{S}(x+y) & x\times\mathsf{S}(y) & \to & (x\times y)+x\\
x+\mathsf{L}(\sigma)) & \to & \mathsf{L}(x+_L\sigma) & x\times\mathsf{L}(\sigma) & \to & \mathsf{L}(x\times_L\sigma)\\
x+_L(y:\sigma) & \to & (x+y):(x+_L\sigma) & x\times_L(y:\sigma) & \to & (x\times y):(x\times_L\sigma)\\
\mathsf{nats}(x) & \to & x:\mathsf{nats}(\mathsf{S}(x)) & \omega & \to & \mathsf{L}(\mathsf{nats}(\mathsf{0}))
\end{array}$$

where $\Delta = \{\mathsf{Ord}\}$ (with Ord a data sort for ordinals) and $\nabla = \{\mathsf{Str}\}$ (with Str a codata sort for streams of ordinals), the ranks for the constructor symbols are: $\mathsf{0} : \mathsf{Ord}$, $\mathsf{S} : \mathsf{Ord} \to \mathsf{Ord}$, $\mathsf{L} : \mathsf{Str} \to \mathsf{Ord}$ and $(:) : \mathsf{Ord}\times\mathsf{Str} \to \mathsf{Str}$. Thus, $\mathcal{C}_\Delta = \{\mathsf{0},\mathsf{S},\mathsf{L}\}$ and we let $\mu_\Delta(\mathsf{S}) = \{1\}$ and $\mu_\Delta(\mathsf{L}) = \varnothing$. For μ given by $\mu(f) = \mu^{can}_\mathcal{R}(f) \cup \mu_\Delta(f)$ for all symbols f, we have $\mu(+) = \mu(+_L) = \mu(\times) = \mu(\times_L) = \{2\}$, $\mu(\mathsf{S}) = \{1\}$, and $\mu(\mathsf{L}) = \mu(:) = \mu(\mathsf{nats}) = \varnothing$. Since $\mathcal{R}$ is μ-terminating (use MU-TERM*), by Theorem 12, $\mathcal{R}$ is productive.*

Endrullis and Hendriks *characterize* productivity as termination of *CSR* [34]. First, an inductively sequential [9] tree specification $\mathcal{R}$ is transformed into a tree specification $\mathcal{R}'$ by a *productivity preserving* transformation. A second transformation yields a CS-TRS $(\mathcal{R}'', \mu)$.

Theorem 13. *Let $\mathcal{R}$ be an inductively sequential tree specification. Then, $\mathcal{R}$ is productive if and only if $\mathcal{R}'$ is productive [34, Theorem 5.5]. And $\mathcal{R}'$ is productive if and only if $\mathcal{R}''$ is μ-terminating [34, Theorem 6.6].*

Although Theorem 12 does *not* provide a characterization of productivity as termination of *CSR* (see [85]), we can use $\mathcal{R}'$ together with Theorem 12 to prove productivity of $\mathcal{R}$ *without using* the second transformation, see [85].

7. Obtaining bounds on runtime complexity

The *derivational height* $\mathsf{dh}(s,\mathsf{R})$ of a term s with respect to a finitely branching relation R is defined (for R-terminating terms s) as the maximal length of R-sequences starting from s [68]. Then, given $n \in \mathbb{N}$, the *derivational complexity* for R is $\mathsf{dc}_\mathsf{R}(n) = \max\{\mathsf{dh}(s,\mathsf{R}) \mid |s| \leq n\}$, where $|s|$ is the *size* of s, i.e., the number of symbols occurring in s. Hence, for all terms s, $\mathsf{dh}(s,\mathsf{R}) \leq \mathsf{dc}_\mathsf{R}(|s|)$. In derivational complexity analysis, we aim at finding (upper and lower) *asymptotic bounds* on $\mathsf{dc}_\mathsf{R}(n)$. Since $\mathsf{dh}(s,\mathsf{R})$ exists for R-terminating terms s only, a well-known technique to obtain bounds on $\mathsf{dc}_\mathsf{R}(n)$ is proving R terminating and then trying to extract such bounds from the *termination technique* which has been used to achieve this goal. For instance, for the term rewriting relation

$\to_{\mathcal{R}}$, bounds on $\mathsf{dc}_{\to_{\mathcal{R}}}(n)$ (or just $\mathsf{dc}_{\mathcal{R}}(n)$) can be obtained from proofs of termination using polynomial interpretations [18, 14, 15, 119], matrix interpretations [101, 106], path orderings [67, 128], dependency pairs [64, 100], etc.

Runtime complexity analysis [64] focuses on obtaining bounds on $\mathsf{dh}(s, \mathsf{R})$ for *basic* terms $s = f(t_1, \ldots, t_k) \in \mathcal{T}_b(\mathcal{F}, \mathcal{X})$, where f is a defined symbol and $t_1, \ldots, t_k$ are constructor terms. Runtime complexity $\mathsf{rc}_{\mathsf{R}}(n)$ is defined as $\mathsf{dc}_{\mathsf{R}}(n)$ but restricting the attention to basic terms s of size $|s| \leq n$ only. Recently, both notions of computational complexity have been related by Fuhs who developed a transformation to obtain bounds on $\mathsf{dc}_{\mathcal{R}}(n)$ from bounds on $\mathsf{rc}_{\mathcal{R}}(n)$ [45].

Hirokawa and Moser use matrix interpretations $\mathcal{A}$ [35] to obtain bounds on $\mathsf{rc}_{\mathsf{R}}(n)$. The interpretation domain is $\mathbb{N}^d$, the set of tuples (or vectors) $\vec{x}$ of d natural numbers. Each k-ary function symbol f is interpreted as a linear expression $f^{\mathcal{A}}(\vec{x}_1, \ldots, \vec{x}_k) = F_1\vec{x}_1 + \cdots + F_k\vec{x}_k + \vec{f_0}$, where $\vec{f_0}$ is a vector of d natural numbers and $F_1, \ldots, F_k$ are d-square matrices of natural numbers. Terms t are interpreted by induction on their structure in the usual way, by using valuation mappings $\alpha : \mathcal{X} \to \mathcal{A}$ to give meaning to variables as follows: (i) $[x]^{\mathcal{A}}_{\alpha} = \alpha(x)$ if $x \in \mathcal{X}$ and (ii) $[f(t_1, \ldots, t_k)]^{\mathcal{A}}_{\alpha} = f^{\mathcal{A}}([t_1]^{\mathcal{A}}_{\alpha}, \ldots, [t_k]^{\mathcal{A}}_{\alpha})$. A (well-founded) ordering $\succ$ on n-tuples of natural numbers is also considered: $\vec{x} \succ \vec{y}$ iff $x_1 > y_1$ and for all $2 \leq i \leq d$, $x_i \geq y_i$. In matrix interpretations, monotonicity of $f^{\mathcal{A}}$ is guaranteed if, for all $1 \leq i \leq ar(f)$, the top leftmost entry $(F_i)_{1,1}$ is positive. We say that $\mathcal{A}$ is compatible with $\mathcal{R}$ if for all $\ell \to r \in \mathcal{R}$ and $\alpha : \mathcal{X} \to \mathcal{A}$, we have $[\ell]^{\mathcal{A}}_{\alpha} \succ [r]^{\mathcal{A}}_{\alpha}$. In *restricted matrix interpretations (RMIs)*, constructor symbols c are interpreted using upper triangular matrices where only 0 or 1 occur in the diagonal entries [65, Section 2]. RMIs permit the obtention of *polynomial* bounds on $\mathsf{rc}_{\mathcal{R}}(n)$. Unfortunately, the monotonicity requirements for compatible RMIs are often difficult to achieve.

Example 19. *Hirokawa and Moser show that no monotone RMI is compatible with the following TRS $\mathcal{R}$ in [65, Example 1] (from [11, Example 2]):*

$$
\begin{array}{rclcrclr}
x - \mathsf{0} & \to & x & (17) & \mathsf{0} \div \mathsf{s}(y) & \to & \mathsf{0} & (19)\\
\mathsf{s}(x) - \mathsf{s}(y) & \to & x - y & (18) & \mathsf{s}(x) \div \mathsf{s}(y) & \to & \mathsf{s}((x-y) \div \mathsf{s}(y)) & (20)
\end{array}
$$

In order to overcome this problem, in [65] Hirokawa and Moser use replacement maps μ to *relax* the monotonicity requirements to μ-monotonicity. In this way, the matrix coefficients F_i in the linear expression for $f^{\mathcal{A}}$ which are required to satisfy $(F_i)_{11} \geq 1$ are those with $i \in \mu(f)$ only.

Example 20. *With $\mu(\mathsf{s}) = \mu(-) = \mu(\div) = \{1\}$, the following 1-dimensional (actually polynomial) interpretation [65, Example 18]*

$$
\mathsf{0}^{\mathcal{A}} = 1 \qquad \mathsf{s}^{\mathcal{A}}(x) = x + 2 \qquad x -^{\mathcal{A}} y = x + 1 \qquad x \div^{\mathcal{A}} y = 3x
$$

is μ-monotonic (but not monotonic; for instance $y > y'$ does not *imply $x -^{\mathcal{A}} y > x -^{\mathcal{A}} y'$). In this way, a linear bound for $\mathsf{rc}_{\mathcal{R}}(n)$ is obtained.*

In order to understand the role of μ-monotonicity to obtain bounds on $\mathsf{rc}_{\mathcal{R}}(n)$, consider $\mathcal{R}$ in Example 19 and μ in Example 20 (where, in particular, $\mu(\div) =$

$\{1\}$). When evaluating a basic term $s = \mathsf{s}(t_1) \div \mathsf{s}(t_2)$, the only applicable rule is (20). After applying it, reducing the first argument $(t_1 - t_2)$ of $\div$ in the obtained reduct $\mathsf{s}((t_1 - t_2) \div \mathsf{s}(t_2))$ could be necessary. However, the second argument $\mathsf{s}(t_2)$ is a constructor term, hence irreducible. Thus, having $2 \notin \mu(\div)$ does not prevent *CSR* from performing necessary reductions. This leads to the notion of *usable replacement map* in runtime complexity which is similar to Fernandez' *usable arguments* (see Section 4.2). Hirokawa and Moser, though, define two kinds of replacement maps: for the runtime analysis of *innermost* rewriting (μ_{i}) and *unrestricted* rewriting (μ_{f}). The definition of μ_{i} is similar to Fernández'; for μ_{f} they use fixpoint techniques. For instance, μ in Example 20 is μ_{f} for $\mathcal{R}$.

For all basic (terminating) terms s, $\mathsf{dh}(s, \to_i)$ is bounded by the maximum length of μ_{i}-rewriting sequences starting from s, i.e., $\mathsf{dh}(s, \to_i) \leq \mathsf{dh}(s, \hookrightarrow_{\mu_{\mathsf{i}}})$; and also $\mathsf{dh}(s, \to) = \mathsf{dh}(s, \hookrightarrow_{\mu_{\mathsf{f}}})$ [65, Corollary 17]. Then, [65, Corollary 20] establishes how μ_{i}-/μ_{f}-monotone RMIs compatible with $\mathcal{R}$ can be used to bound the (innermost) runtime complexity of $\mathcal{R}$: for M the component-wise maximum of *all matrices* C_i, $1 \leq i \leq k$ used in the interpretation $c^{\mathcal{A}}(x_1, \ldots, x_k) = \sum_{i=1}^{k} C_i x_i + \vec{c}_0$ of constructor symbols $c \in \mathcal{C}$, the number p of *ones* occurring along the diagonal of M yields the polynomial bound $O(n^p)$.

Remark 5 (Runtime complexity bounds for *CSR*).$^{(\star)}$ *As a matter of fact, Hirokawa and Moser's work provides the* first analysis of runtime complexity *of CSR. Indeed, a μ-monotonic RMI $\mathcal{A}$ compatible with a TRS $\mathcal{R}$ proves μ-termination of $\mathcal{R}$. The validity of the polynomial bounds obtained from matrix interpretations $\mathcal{A}$ in [65, Section 2] does not depend on any monotonicity assumption. Thus, given a replacement map μ, they actually provide bounds on $\mathsf{rc}_{\mathcal{R},\mu}(n) = \max\{\mathsf{dh}(s, \hookrightarrow_\mu) \mid s \in \mathcal{T}_b(\mathcal{F}, \mathcal{X}), |s| \leq n\}$, the runtime complexity bound for* CSR*, which then bounds $\mathsf{rc}_{\mathcal{R}}(n)$ (resp. $\mathsf{rc}^i_{\mathcal{R}}(n)$) as a consequence of $\mathsf{dh}(s, \to) = \mathsf{dh}(s, \hookrightarrow_{\mu_{\mathsf{f}}})$ if $\mu = \mu_{\mathsf{f}}$ (resp. $\mathsf{dh}(s, \to_i) \leq \mathsf{dh}(s, \hookrightarrow_{\mu_{\mathsf{i}}})$ if $\mu = \mu_{\mathsf{i}}$). Therefore, the results in [65, Section 2] can be used to obtain bounds on $\mathsf{rc}_{\mathcal{R},\mu}(n)$ for arbitrary replacement maps μ.*

In [76] similar ideas are developed to provide the first complexity analysis for *conditional* TRSs by also relying on transformations of CTRSs into CS-TRSs.

8. *CSR* for variants of term rewriting

In order to make the advantages of *CSR* available in other computational settings and programming languages, several *extensions* to more general rewriting-based frameworks like *conditional* rewriting, *constrained* rewriting (where the rules include a *conditional part* to be tested before being able to rewrite any call), *equational rewriting* (where terms are rewritten *modulo* an equational theory), and *narrowing* (where pattern matching is replaced by *unification* when function calls are bound to rules) have been envisaged.

Remark 6 (Introducing context-sensitivity). *Rule $(C)_{f,i}$ in Figure 2 often occur in inference systems $\mathcal{I}$. A systematic way to make $\mathcal{I}$ 'context-sensitive' is using $(C)_{f,i}$ for all $i \in \mu(f)$ rather than for all $1 \leq i \leq ar(f)$.*

In this section, we briefly discuss some known extensions, which we often recast as a simple transformation of inference systems following Remark 6.

8.1. Conditional context-sensitive rewriting

By a CS-CTRS we mean a pair $(\mathcal{R},\mu)$ where $\mathcal{R} = (\mathcal{F}, R)$ is a CTRS and $\mu \in M_{\mathcal{F}}$. In [26], *CSR* was extended to *CTRSs* by applying the procedure in Remark 6 to obtain

$$\mathfrak{I}_{\text{CS-CTRS}}[\mathbb{S},\mathbb{M},\mathbb{R}] = \{(\text{Rf}),(\text{T})\}\cup\{(\text{C})_{f,i} \mid f\in\mathbb{S}, i\in\mathbb{M}(f)\}\cup\{(\text{CRl})_\alpha \mid \alpha\in\mathbb{R}\}.$$

An inference system $\mathcal{I}(\mathcal{R},\mu) = \mathfrak{I}_{\text{CS-CTRS}}[\mathcal{F},\mu,\mathcal{R}]$ for a CTRS $\mathcal{R} = (\mathcal{F}, R)$ and $\mu \in M_{\mathcal{R}}$ is obtained and one-step and many-step conditional μ-rewriting $\hookrightarrow_\mu$ and $\hookrightarrow^*_\mu$ are defined as provability of goals $s \to t$ and $s \to^* t$ in $\mathcal{I}(\mathcal{R},\mu)$.

8.1.1. Operational termination of CS-CTRSs

We say that a CTRS $\mathcal{R}$ is operationally μ-terminating if there are no terms s and t with an infinite proof tree for $s \to^* t$ in $\mathcal{I}(\mathcal{R},\mu)$. Operational termination of CS-CTRSs was investigated in [26] by using transformation $\mathcal{U}_{opt}$ (see Section 5.1) with μ extended to symbols U_i^α as before (denoted $\overline{\mu}^{\mathcal{U}}$):

$$\overline{\mu}^{\mathcal{U}}(f) \quad = \quad \begin{cases} \mu(f) & \text{if } f\in\mathcal{F} \\ \{1\} & \text{otherwise, i.e., for symbols } U_i^\alpha \end{cases}$$

We have the following

Theorem 14. *Let $\mathcal{R}$ be a DCTRS and $\mu \in M_{\mathcal{R}}$.*

1. *[23, Lemma 3] If $s \hookrightarrow_{\mathcal{R},\mu} t$, then $s \hookrightarrow^*_{\mathcal{U}_{opt}(\mathcal{R}),\overline{\mu}^{\mathcal{U}}} t$.*
2. *[23, Theorem 2] If $\mathcal{U}_{opt}(\mathcal{R})$ is $\overline{\mu}^{\mathcal{U}}$-terminating, then $\mathcal{R}$ is operationally μ-terminating.*

The analysis of operational termination of CS-CTRSs by using dependency pairs (as done for CTRSs [90, 91, 92] and CS-TRSs [2, 58]) is still a subject for future work.

8.1.2. Confluence of CS-CTRSs

A CS-CTRS $(\mathcal{R},\mu)$ is μ-confluent if $\hookrightarrow_{\mathcal{R},\mu}$ is confluent. We have the following generalization of Theorem 8 (regarding confluence).

Theorem 15.$^{(\star)}$ *Let $\mathcal{R}$ be a DCTRS and $\mu \in M_{\mathcal{R}}$. If $\mathcal{R}$ is μ-terminating and $\mathcal{U}_{opt}(\mathcal{R})$ is $\overline{\mu}^{\mathcal{U}}$-confluent, then $\mathcal{R}$ is μ-confluent.*

Proof. By contradiction. If $\mathcal{R}$ is not μ-confluent, then there are terms $s,t,t' \in \mathcal{T}(\mathcal{F},\mathcal{X})$ such that $s \hookrightarrow^*_{\mathcal{R},\mu} t$ and $s \hookrightarrow^*_{\mathcal{R},\mu} t'$ but t and t' are not $\hookrightarrow^*_{\mathcal{R},\mu}$-joinable. By μ-termination of $\mathcal{R}$, we can assume that t and t' are different *μ-normal forms* $t \neq t'$. By Theorem 14.(1), $s \hookrightarrow^*_{\mathcal{U}_{opt}(\mathcal{R}),\overline{\mu}^{\mathcal{U}}} t$ and $s \hookrightarrow^*_{\mathcal{U}_{opt}(\mathcal{R}),\overline{\mu}^{\mathcal{U}}} t'$. By $\overline{\mu}^{\mathcal{U}}$-confluence of $\mathcal{U}_{opt}(\mathcal{R})$, there is u such that $t \hookrightarrow^*_{\mathcal{U}_{opt}(\mathcal{R}),\overline{\mu}^{\mathcal{U}}} u$ and $t' \hookrightarrow^*_{\mathcal{U}_{opt}(\mathcal{R}),\overline{\mu}^{\mathcal{U}}} u$. However, since t and t' are μ-normal forms built from variables and symbols in $\mathcal{F}$ only, they are $\hookrightarrow_{\mathcal{U}_{opt}(\mathcal{R}),\overline{\mu}^{\mathcal{U}}}$-normal forms and $t = u = t'$, a contradiction. $\square$

Example 21. *The following CTRS* $\mathcal{R}$

$$\begin{array}{rclr}
\mathsf{a} & \to & \mathsf{b} & (21)\\
\mathsf{f}(x) & \to & y \Leftarrow \mathsf{g}(x) \to \mathsf{c}(y) & (22)
\end{array}
\qquad
\begin{array}{rclr}
\mathsf{g}(\mathsf{a}) & \to & \mathsf{c}(\mathsf{d}) & (23)\\
\mathsf{g}(\mathsf{b}) & \to & \mathsf{c}(\mathsf{b}) & (24)
\end{array}$$

is not *confluent. We have* $\underline{\mathsf{f}(\mathsf{a})} \to_{\mathcal{R}} \mathsf{d}$ *because* $\underline{\mathsf{g}(\mathsf{a})} \to_{\mathcal{R}} \mathsf{c}(\mathsf{b})$ *using (23). Also,* $\mathsf{f}(\underline{\mathsf{a}}) \to_{\mathcal{R}} \underline{\mathsf{f}(\mathsf{b})} \to_{\mathcal{R}} \mathsf{b}$ *because* $\underline{\mathsf{g}(\mathsf{b})} \to_{\mathcal{R}} \mathsf{c}(\mathsf{d})$ *using (24). Now, consider the replacement map* $\mu(\mathsf{f}) = \mu(\mathsf{g}) = \varnothing$ *and* $\mu(\mathsf{c}) = \{1\}$*. The TRS* $\mathcal{U}_{opt}(\mathcal{R})$ *is*

$$\begin{array}{rcl}
\mathsf{a} & \to & \mathsf{b}\\
\mathsf{f}(x) & \to & U(\mathsf{g}(x))\\
U(\mathsf{c}(y)) & \to & y
\end{array}
\qquad
\begin{array}{rcl}
\mathsf{g}(\mathsf{a}) & \to & \mathsf{c}(\mathsf{d})\\
\mathsf{g}(\mathsf{b}) & \to & \mathsf{c}(\mathsf{b})
\end{array}$$

and $\overline{\mu}^{\mathcal{U}}$ *is* $\overline{\mu}^{\mathcal{U}}(\mathsf{f}) = \overline{\mu}^{\mathcal{U}}(\mathsf{g}) = \varnothing$ *and* $\overline{\mu}^{\mathcal{U}}(\mathsf{c}) = \overline{\mu}^{\mathcal{U}}(U) = \{1\}$*. There is no* $\overline{\mu}^{\mathcal{U}}$*-critical pair.*[18] *Furthermore,* $\mathcal{U}_{opt}(\mathcal{R})$ *has* left-homogeneous μ-replacing variables *(μ-*LHRV*), i.e., each active variable in the left-hand side of a rule is active everywhere in the rule [86, Section 8.1]. According to [86, Sections 8.3 and 8.4]* $\mathcal{U}_{opt}(\mathcal{R})$ *is locally* $\overline{\mu}^{\mathcal{U}}$*-confluent and hence* $\overline{\mu}^{\mathcal{U}}$*-confluent due to* $\overline{\mu}^{\mathcal{U}}$*-termination of* $\mathcal{U}_{opt}(\mathcal{R})$ *(use* MU-TERM*). By Theorem 15,* $\mathcal{R}$ *is* μ*-confluent.*

Unfortunately, μ-confluence of CTRSs is underexplored to date. However, [27, Definition 10] generalizes the notions of μ-critical pair and conditional critical pair for CTRSs[19] to define *conditional critical pairs* for order-sorted conditional rewrite theories with frozenness specifications φ (see Section 3) and [27, Theorem 5] uses them to prove *coherence* of such rewrite theories.

8.1.3. Canonical CSR *with CS-CTRSs*

Regarding *canonical CSR* (Section 3), when dealing with (oriented) CS-CTRSs, we need to revise the notion of canonical replacement map to consider the reachability goals $s_i \to^* t_i$ in $(\mathrm{CRl})^{\mathcal{R}}_{\rho}$ as μ-reachability problems $\sigma(s_i) \hookrightarrow^*_{\mathcal{R},\mu} \sigma(t_i)$. The canonical replacement map $\mu^{can}_{\mathcal{R}}$ for CTRSs $\mathcal{R}$ is

> *the most restrictive replacement map* μ *ensuring that, for all* $\ell \to r \Leftarrow s_1 \approx t_1, \ldots, s_n \approx t_n \in \mathcal{R}$, $\mathcal{P}os_{\mathcal{F}}(\ell) \subseteq \mathcal{P}os^{\mu}(\ell)$ *and* $\mathcal{P}os_{\mathcal{F}}(t_i) \subseteq \mathcal{P}os^{\mu}(t_i)$, $1 \leq i \leq n$ *[84, Section 3.1].*

This does not suffice, though: the μ-normal forms of left-linear TRSs are head-normal forms if $\mu \in CM_{\mathcal{R}}$ [86, Section 6]; the ability of *CSR* to compute canonical forms relies on this fact [86, Section 9]. This fails to hold for CTRSs.

Example 22. *Let* $\mathcal{R} = \{\mathsf{a} \to \mathsf{b}, \mathsf{f}(x) \to \mathsf{b} \Leftarrow \mathsf{b} \to x\}$*. Since* $1 \notin \mu^{can}_{\mathcal{R}}(\mathsf{f})$ *and* $\mathsf{b} \not\hookrightarrow^*_{\mu} \mathsf{a}$*, term* $\mathsf{f}(\mathsf{a})$ *is not* $\mu^{can}_{\mathcal{R}}$*-reducible. However,* $\mathsf{f}(\underline{\mathsf{a}}) \to \mathsf{f}(\mathsf{b})$ *and* $\mathsf{f}(\mathsf{b})$ *is a redex, i.e.,* $\mathsf{f}(\mathsf{a})$ *is* not *a head-normal form.*

[18] A μ-critical pair is a critical pair whose critical position is active, see [86, Section 8.2].

[19] For CTRSs we have *conditional* critical pairs $\langle s, t\rangle \Leftarrow c$, where the conditional part c is empty if both rules defining the critical pair are unconditional, see [116, Definition 7.1.8].

Recall from Section 5.3 that *rhs*'s t_i of conditions $s_i \to t_i$ in rules of *normal* CTRSs are *ground* and contain no preredex (see footnote 16). For left-linear and normal CTRSs, if μ is canonical, then μ-normal forms are head-normal forms [84, Theorem 3]. Note that $\mathcal{R}$ in Example 22 is *not* normal.

8.2. Built-in numbers and collection data structures

Falke and Kapur integrate replacement restrictions into their *Constrained Equational Rewrite Systems* (CERSs [37]) that extend TRSs with built-in data structures, in particular integer numbers and collection data structures. In CERSs, constrained rules $\ell \to r\,[\![\varphi]\!]$ are allowed. Here φ is a numeric constraint, i.e., a Boolean expression with atoms $s \bowtie t$ where $\bowtie \in \{>, \geq, \simeq\}$. Such formulas φ are handled *apart*, as 'built-ins', and the rewrite relation is *not* used in satisfiability tests [37, Definition 5]. Computations with CERSs can be described using the generic system

$$\mathfrak{I}_{\mathrm{CERS}}[\mathbb{S},\mathbb{R}] = \{(\mathrm{R}),(\mathrm{T})\} \cup \{(\mathrm{C})_{f,i} \mid f \in \mathbb{S}, 1 \leq i \leq ar(f)\} \cup \{(\mathrm{CERl})_\alpha \mid \alpha \in \mathbb{R}\}$$

where $(\mathrm{CERl})_\alpha$ is $\frac{\varphi}{\ell \to r}$ for $\alpha : \ell \to r\,[\![\varphi]\!]$. Proofs of φ are assumed to be derived to the appropriate subsytem. Falke and Kapur use *CSR* to avoid infinite computations when dealing with infinite data structures such as sets of integers, etc. This enables a more natural specification of some algorithms in the rewriting framework [38]. The integration of CS replacement restrictions in CERSs follows the usual approach of allowing reductions on μ-replacing positions only [37, Definition 7]. Equivalently, this corresponds to rely on

$$\mathfrak{I}_{\text{CS-CERS}}[\mathbb{S},\mathbb{M},\mathbb{R}] = \{(\mathrm{R}),(\mathrm{T})\} \cup \{(\mathrm{C})_{f,i} \mid f \in \mathbb{S}, i \in \mathbb{M}(f)\} \cup \{(\mathrm{CERl})_\alpha \mid \alpha \in \mathbb{R}\}.$$

The authors extend the dependency pair framework in [1] to CERSs, thus being able to prove termination of CS-CERS as well [36, 38].

8.3. Context-sensitive rewriting modulo

The generic inference system

$$\mathfrak{I}_{\mathrm{ETRS}}[\mathbb{S},\mathbb{E},\mathbb{R}] = \mathfrak{I}_{\mathrm{EQ}}[\mathbb{S},\mathbb{E}] \cup \{(\mathrm{C})_{f,i} \mid f \in \mathbb{S}, 1 \leq i \leq ar(f)\} \cup \{(\mathrm{Rl})_\alpha \mid \alpha \in \mathbb{R}\} \cup \mathfrak{I}_{\mathrm{RM}}$$

where

$$\mathfrak{I}_{\mathrm{EQ}}[\mathbb{S},\mathbb{E}] = \{(\mathrm{ER}),(\mathrm{ET})\} \cup \{(\mathrm{EC})_{f,i} \mid f \in \mathbb{S}, 1 \leq i \leq ar(f)\} \cup \{(\mathrm{Eq})_\varepsilon \mid \varepsilon \in \mathbb{E}\}$$

is the generic system for equational deduction with a set of equations $\mathcal{E}$ (see the inference rules in Figure 4) and $\mathfrak{I}_{\mathrm{RM}} = \{(\mathrm{MR}),(\mathrm{MT}),(\mathrm{RM})\}$ encodes rewriting modulo with $\mathcal{R}$ (Figure 5), can be used to define *one-step* and *many-step* rewritings $\to_{\mathcal{R}/\mathcal{E}}$ and $\to^*_{\mathcal{R}/\mathcal{E}}$ with a TRS $\mathcal{R}$ *modulo a set of equations* $\mathcal{E}$ as provability of goals $s \to_= t$ and $s \to^*_= t$ in an inference system $\mathcal{I}(\mathcal{R},\mathcal{E}) = \mathfrak{I}_{\mathrm{ETRS}}[\mathcal{F},\mathcal{E},\mathcal{R}]$.

Context-sensitive rewriting modulo *associativity* and *commutativity* was first considered in [40]. The authors use a *restricted* equational theory generated by

$$\text{(ER)}\quad \frac{}{x = x} \qquad \text{(EC)}_{f,i}\quad \frac{x_i = y_i}{f(x_1,\ldots,x_i,\ldots,x_k) = f(x_1,\ldots,y_i,\ldots,x_k)}$$

$$\text{(ET)}\quad \frac{x = y \qquad y = z}{x = z} \qquad \text{(Eq)}_{s=t}\quad \frac{}{s = t}$$

Figure 4: Parametric inference rules for equational reasoning

$$\text{(MR)}\quad \frac{x = y}{x \to_{=}^{*} y} \qquad \text{(RM)}\quad \frac{w = x \quad x \to y \quad y = z}{w \to_{=} z} \qquad \text{(MT)}\quad \frac{x \to_{=} y \qquad y \to_{=}^{*} z}{x \to_{=}^{*} z}$$

Figure 5: Parametric inference rules for rewriting modulo

a set of equations and a replacement map μ [40, Definition 6]. This is equivalent to use

$$\mathfrak{I}_{\text{CS-EQ}}[\mathbb{S},\mathbb{M},\mathbb{E}] = \{(\text{ER}),(\text{ET})\} \cup \{(\text{EC})_{f,i} \mid f \in \mathbb{S}, i \in \mathbb{M}(f)\} \cup \{(\text{Eq})_{\varepsilon} \mid \varepsilon \in \mathbb{E}\},$$

which is like $\mathfrak{I}_{\text{EQ}}$ with the range of i controlled by a replacement map in $(\text{EC})_{f,i}$. For the context-sensitive extension of rewriting modulo, Ferreira and Ribeiro use a *second* replacement map μ'. Then, [40, Definition 7] corresponds to the use of

$$\begin{array}{rcl} \mathfrak{I}_{\text{CS-ETRS}}[\mathbb{S},\mathbb{M}_1,\mathbb{E},\mathbb{M}_2,\mathbb{R}] & = & \mathfrak{I}_{\text{CS-EQ}}[\mathbb{S},\mathbb{M}_1,\mathbb{E}] \cup \{(\text{C})_{f,i} \mid f \in \mathbb{S}, i \in \mathbb{M}_2(f)\} \\ & & \cup\, \{(\text{Rl})_{\alpha} \mid \alpha \in \mathbb{R}\} \cup\, \mathfrak{I}_{\text{RM}} \end{array}$$

to obtain an inference system $\mathcal{I}(\mathcal{E},\mathcal{R},\mu,\mu') = \mathfrak{I}_{\text{CS-ETRS}}[\mathcal{F},\mu,\mathcal{E},\mu',\mathcal{R}]$ in proofs of goals $s \to_{=} t$ and $s \to_{=}^{*} t$ (one-step and many step context-sensitive rewriting modulo, respectively). Note the use of μ (with $(\text{EC})_{f,i}$, as part of $\mathfrak{I}_{\text{CS-EQ}}[\mathcal{F},\mu,\mathcal{E}]$) and μ' (with $(C)_{f,i}$).

For termination analysis, Ferreiro and Ribeiro consider AC-theories where $\mathcal{E}$ only contains *associative* and *commutative* axioms $f(x,f(y,z)) = f(f(x,y),z)$ and $f(x,y) = f(y,x)$, for each AC (binary) symbol f. An *AC-rewrite system* (denoted $\mathcal{R}/AC$) is an equational rewrite system $\mathcal{R}/\mathcal{E}$ where $\mathcal{E}$ is an AC-theory. They restrict the rewriting steps with μ_r, and define a restricted equational theory with μ_{ac} (i.e., $\mu = \mu_{ac}$ and $\mu' = \mu_r$ above). They characterize termination of AC-*CSR* using orderings [40, Theorem 2].

Example 23. *Consider the following TRS [40, Example 4]*

$$\begin{array}{rcl@{\qquad\qquad}rcl} \mathsf{It}(p) & \to & p\,;\mathsf{It}(p) & p+q & \to & p \\ (p+q)\,;r & \to & (p\,;r)+(q\,;r) & p+q & \to & q \\ (p\,;q)\,;r & \to & p\,;(q\,;r) & p \parallel \mathsf{abort} & \to & \mathsf{abort} \\ p\,;\mathsf{skip} & \to & p & p \parallel \mathsf{skip} & \to & p \\ \mathsf{abort}\,;p & \to & \mathsf{abort} & p \parallel (q+r) & \to & (p \parallel q)+(p \parallel r) \end{array}$$

for a process language with parallel composition ($\parallel$) and choice ($+$) AC operators. For the sequential composition operator ($;$) we have $\mu_r(;) = \{1\}$. No further replacement restrictions are imposed with μ_r. And no replacement map

μ_{ac} *is considered here. Termination of* $\hookrightarrow_{\mathcal{R}/AC,\mu_r}$ *is proved by the following polynomial interpretation: the domain is* $\mathcal{A} = \mathbb{N}_2$*, i.e., the set of natural numbers bigger than* 1*; for function symbols:*

$$\begin{array}{ccc} \mathsf{abort}^{\mathcal{A}} = 2 & \mathsf{skip}^{\mathcal{A}} = 2 & \mathsf{It}^{\mathcal{A}}(x) = 2x+1 \\ x\ ;^{\mathcal{A}}\ y = 2x & x +^{\mathcal{A}} y = x+y+1 & x \parallel^{\mathcal{A}} y = xy \end{array}$$

Note that $+^{\mathcal{A}}$ *and* $\parallel^{\mathcal{A}}$ *are commutative and associative. Also note that* $;^{\mathcal{A}}$ *is* μ_r*-monotonic (but not monotonic). The key point for proving AC* μ_r*-termination of* $\mathcal{R}$ *is compatibility of the ordering* $>$ *over the naturals with the first rule of the TRS, i.e.,* $\mathsf{It}(p) \to p\ ;\ \mathsf{It}(p)$*. We develop this here: for all* $p \in \mathcal{A}$*,*

$$\mathsf{It}(p)^{\mathcal{A}} = 2p+1 > 2p = (p\ ;\ \mathsf{It}(p))^{\mathcal{A}}$$

Ferreira and Ribeiro also introduced a transformation for proving termination of AC-*CSR* by proving AC-termination of the obtained TRS. Following Ferreira and Ribeiro's work, Giesl and Middeldorp developed new and more powerful transformations for proving termination of AC-*CSR* [49]. The analysis of termination of AC-CS-TRSs by using dependency pairs (as done for CS-TRSs (see references above) and AC-TRSs [4, 129]) is still a subject for future work.

8.4. Context-sensitive narrowing

Functional Logic Languages integrate the most interesting features of pure logic and functional languages in a unified framework [10]. Logical variables, partial data structures and search for solutions (from the logic programming side), are available in functional logic languages. Nested expressions, higher-order functions and the possibility of benefitting from the deterministic nature of functions also become available from the functional component [61, 62]. In order to deal with logic variables we need an operational mechanism to instantiate them during the evaluation of expressions. This mechanism is *narrowing*, which combines term rewriting and unification [126]. A term s narrows to t, written $s \leadsto_{[p,\alpha,\sigma]} t$, if there is $p \in \mathcal{P}os_{\mathcal{F}}(s)$ and a variant (i.e., a renamed version) of a rule $\alpha : \ell \to r$ such that $s|_p$ and ℓ unify with (idempotent) mgu σ, and $t = \sigma(s[r]_p)$. The idea of limiting narrowing by means of replacement restrictions is considered in [79, Section 6].

Definition 4 (Context-sensitive narrowing). [79, Definition 15] *Let* $\mathcal{R}$ *be a TRS and* $\mu \in M_{\mathcal{R}}$*. A term* s μ*-narrows to* t *(*$s \overset{\mu}{\leadsto}_{[p,\alpha,\sigma]} t$*) if* $s \leadsto_{[p,\alpha,\sigma]} t$ *and* $p \in \mathcal{P}os^{\mu}(t)$*.*

Context-sensitive narrowing can be used in Maude 3.0.

Example 24. *The* Maude *module in Figure 6 encodes the TRS* $\mathcal{R}$ *in [79, Example 1] for use in narrowing computations with* Maude *3.0 (hence the mandatory labels* `[narrowing]` *in each rule). When* `if(and(x,ff),y + s(0),0)` *is narrowed, the second argument* `y + s(0)` *of* `if` *should be narrowed only after being appropriately instantiatiated and having evaluated the condition* `and(x,ff)`*. With the frozenness annotation* φ *given by*

```
mod Ex1_JFLP98 is
  sort S .
  ops tt ff 0 : -> S .
  op s : S -> S .
  op if : S S S -> S [frozen (2 3)] .
  ops and _+_ : S S -> S .
  var x y : S .
  rl if(tt,x,y) => x [narrowing] .
  rl if(ff,x,y) => y [narrowing] .
  rl and(tt,x) => x [narrowing] .
  rl and(ff,x) => ff [narrowing].
  rl 0 + x =>  x [narrowing] .
  rl s(x) + y => s(x + y) [narrowing] .
endm
```

Figure 6: Context-sensitive narrowing in Maude

```
  op if : S S S -> S [frozen (2 3)] .
```

(i.e., $\mu_\varphi(\texttt{if}) = \{1\}$*) we obtain the desired effect by using the narrowing command* `vu-narrow` *of* Maude *3.0 [22, Section 7], with the expression*

```
  if(and(x:S,ff),y:S + s(0),0) =>! Z:S
```

This computes all possible narrowings of the expression. The different outcoming values are kept in variable `Z`*, whilst* `x` *and* `y` *are instantiated at need, see Figure 7. Only two narrowing evaluation sequences are obtained, for the two instantiations of* `x` *to either* `tt` *or* `ff`*, when using the* `and`*-rules. No 'real' instantiation of* `y` *is attempted to (wastefully) evaluate the second argument, as it is forbidden by* φ*. Actually, the attempt to narrow* `if(and(x,ff),y + s(0),0)` *with no* `frozen` *annotation in* `Ex1_JFLP98` *leads to an* infinite computation *with* `Z` *always instantiated to* `0`*, but* `y` *instantiated to* `0`*,* `s(0)`*,* `s(s(0))`*, ...*

9. Variants of *CSR* for term rewriting

Several authors have devised *weaker* restrictions of rewriting trying to improve the computational power of *CSR*. As discussed in [86, Section 11], the canonical replacement map $\mu^{can}_{\mathcal{R}}$ is the usual starting point to use *CSR* in computations with a (left-linear) TRS $\mathcal{R}$. This guarantees that a number of results and techniques can be used to perform semantically meaningful computations, see [86, Section 9]. However, termination of *CSR* is often an important reason to use *CSR* instead of (a strategy for) unrestricted rewriting, see [86, Remark 1.2]. In some cases, though, $\mu^{can}_{\mathcal{R}}$ fails to achieve both requirements.

```
Maude> vu-narrow in Ex1_JFLP98 : if(and(x:S,ff),y:S + s(0),0) =>! Z:S .
vu-narrow in Ex1_JFLP98 : if(and(x, ff), y + s(0), 0) =>! Z:S .

Solution 1
rewrites: 4 in 0ms cpu (1ms real) (4305 rewrites/second)
state: 0
accumulated substitution:
x --> tt
y --> %1:S
variant unifier:
Z:S --> 0

Solution 2
rewrites: 4 in 1ms cpu (3ms real) (3350 rewrites/second)
state: 0
accumulated substitution:
x --> ff
y --> %1:S
variant unifier:
Z:S --> 0

No more solutions.
rewrites: 4 in 1ms cpu (3ms real) (3254 rewrites/second)
```

Figure 7: A narrowing evaluation in Maude

Example 25. *Consider the following TRS $\mathcal{R}$ from [104]*

$$\begin{array}{rcl} \mathsf{2nd}(x:y:z) & \to & y \\ \mathsf{from}(x) & \to & x:\mathsf{from}(\mathsf{s}(x)) \end{array} \tag{25}$$

Since $\mu^{can}_{\mathcal{R}}(\mathsf{from}) = \mu^{can}_{\mathcal{R}}(\mathsf{s}) = \varnothing$, $\mu^{can}_{\mathcal{R}}(\mathsf{2nd}) = \{1\}$ *and* $\mu^{can}_{\mathcal{R}}(:) = \{2\}$, CSR *obtains the value of* $s = \mathsf{2nd}(\mathsf{from}(0))$*:*

$$\begin{array}{rcl} \mathsf{2nd}(\underline{\mathsf{from}(0)}) & \hookrightarrow_{\mu^{can}_{\mathcal{R}}} & \mathsf{2nd}(0:\underline{\mathsf{from}(\mathsf{s}(0))}) \\ & \hookrightarrow_{\mu^{can}_{\mathcal{R}}} & \underline{\mathsf{2nd}(0:\mathsf{s}(0):\mathsf{from}(\mathsf{s}(\mathsf{s}(0))))} \hookrightarrow_{\mu^{can}_{\mathcal{R}}} \mathsf{s}(0) \end{array} \tag{26}$$

Unfortunately, $\mathcal{R}$ *is not* $\mu^{can}_{\mathcal{R}}$*-terminating due to (25). An 'eager' attempt to evaluate* $\mathsf{from}(\mathsf{s}(\mathsf{s}(0)))$ *in the last step of the sequence before applying the rule for* 2nd *leads to an infinite computation:*

$$\begin{array}{rcl} \mathsf{2nd}(\underline{\mathsf{from}(0)}) & \hookrightarrow_{\mu^{can}_{\mathcal{R}}} & \mathsf{2nd}(0:\underline{\mathsf{from}(s(0))}) \\ & \hookrightarrow_{\mu^{can}_{\mathcal{R}}} & \mathsf{2nd}(0:\mathsf{s}(0):\underline{\mathsf{from}(\mathsf{s}(\mathsf{s}(0)))}) \hookrightarrow_{\mu^{can}_{\mathcal{R}}} \cdots \end{array} \tag{27}$$

With $\mu(:) = \varnothing$*, though,* $\mathcal{R}$ *is* μ*-terminating but the evaluation stops too early*

$$\mathsf{2nd}(\underline{\mathsf{from}(0)}) \hookrightarrow_{\mu} \mathsf{2nd}(0:\mathsf{from}(\mathsf{s}(0)))$$

Note that $\mu \notin CM_{\mathcal{R}}$*. The obtained* μ*-normal form* $u = \mathsf{2nd}(0:\mathsf{from}(\mathsf{s}(0)))$ *is* not *a head-normal form. Thus, it is* not *safe to jump into the maximal frozen part* $\mathsf{from}(\mathsf{s}(0)$ *to perform* normalization via-μ-normalization *(see Figure 3) of* u *as this would lead to an infinite sequence again.*

ROOT	$\Lambda \in \mathcal{P}os^{\gamma}(t)$
DOWN	$p.q \in \mathcal{P}os^{\gamma}(C[t]_p) \Rightarrow p \in \mathcal{P}os^{\gamma}(C[t]_p)$
SUBTERM	$p.q \in \mathcal{P}os^{\gamma}(C[t]_p) \Rightarrow q \in \mathcal{P}os^{\gamma}(t)$
COMP	$(p \in \mathcal{P}os^{\gamma}(C[t]_p) \wedge q \in \mathcal{P}os^{\gamma}(t)) \Rightarrow p.q \in \mathcal{P}os^{\gamma}(C[t]_p)$

Figure 8: A list of basic properties of syntactic replacement restrictions, where $C, t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$

Example 25 shows how the usual computational procedures based on *CSR* may fail. In the following, we consider some alternative mechanisms that have been proposed to overcome these problems. We often illustrate them in connection with Example 25 to see how do they help to solve such problems.

9.1. A more general framework for syntactic replacement restrictions

The replacement restrictions on terms t induced by a replacement map μ (i.e., $\mathcal{P}os^{\mu}(t)$) are a particular case of a more general notion of *syntactic replacement restriction* which *directly* focuses on the *positions* of terms [78].

Definition 5 (Syntactic replacement restriction). *Let $\mathcal{F}$ be a signature. A syntactic replacement restriction γ is a mapping $\gamma : \mathcal{T}(\mathcal{F}, \mathcal{X}) \to \wp(N^*_{>0})$ such that, for all $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, $\gamma(t) \subseteq \mathcal{P}os(t)$.*

We often write $\mathcal{P}os^{\gamma}(t)$ rather than $\gamma(t)$. Let $\Gamma_{\mathcal{F}}$ (or just Γ) be the set of syntactic replacement restrictions for $\mathcal{F}$ and Γ_{csr} be the subset of Γ induced by replacement maps $\mu \in M_{\mathcal{F}}$. Many replacement restrictions can be associated to a given signature. We identify a property PROP on replacement restrictions with a subset $\Gamma^{\mathsf{PROP}} \subseteq \Gamma$ of Γ and say that γ *has property* PROP if $\gamma \in \Gamma^{\mathsf{PROP}}$. Figure 8 shows some basic properties of replacement restrictions:

- ROOT enables reductions of a term at the root.
- DOWN guarantees that $\mathcal{P}os^{\gamma}(t)$ is *prefix closed*. This is useful to implement systems using these restrictions: when looking inside a term t for a (replacing) redex, if we find a nonreplacing position $p \in \overline{\mathcal{P}os^{\gamma}}(t)$, we can stop the search for other redexes below: they are nonreplacing as well.
- $\mathsf{SUBTERM}$ and COMP concern *locality* of (restricted) computations: after replacing a redex we can *locally* resume the search for a new (replacing) redex; no backtracking to the root of the maximal input term is necessary.

Given properties $\mathsf{PROP_1}, \ldots, \mathsf{PROP_n}$, $\Gamma^{\wedge_{i=1}^{n}\mathsf{PROP_i}} = \bigcap_{i=1}^{n} \Gamma^{\mathsf{PROP_i}}$ collects all restritions which simultaneously satisfy $\mathsf{PROP_1}, \ldots, \mathsf{PROP_n}$. We have the following characterization of context-sensitive replacement restrictions.

Theorem 16. [78] $\Gamma_{csr} = \Gamma^{\mathsf{ROOT} \wedge \mathsf{COMP} \wedge \mathsf{DOWN} \wedge \mathsf{SUBTERM}}$.

The variants of *CSR* described below can be seen as appropriate ways to specify and analyze syntactic replacement restrictions.

```
mod! TEST {
  [T]
  op 0    : -> T
  op s    : T -> T        {strat: (1)}
  op _:_  : T T -> T      {strat: (1 -2)}
  op 2nd  : T   -> T      {strat: (1 0)}
  op from : T   -> T      {strat: (0)}
  vars X Y Z : T
  eq from(X) = X:from(s(X)) .
  eq 2nd(X:(Y:Z)) = Y .
}
```

Figure 9: CafeOBJ program with on-demand strategy annotations

9.2. On-demand strategy annotations

On-demand E-strategies are sequences $\xi(f) = (i_1 \cdots i_n)$ of *integers* associated to k-ary function symbols f in CafeOBJ programs, so that $-k \leq i_j \leq k$ for all $1 \leq j \leq n$. They guide the *evaluation strategy* of function calls $f(t_1, \ldots, t_k)$ by taking indices i from $\xi(f)$ from left-to-right and: (a) if $i > 0$, then t_i is (recursively) evaluated; (b) if $i = 0$, a rule defining f is attempted; and (c) if $i < 0$, then $t_{|i|}$ is evaluated 'on-demand', where a 'demand' is an attempt to match a pattern against $t_{|i|}$ [103, 104, 114]. Negative annotations aim at avoiding nontermination while complete evaluations of expressions are still possible.

Example 26. *The* CafeOBJ *program in Figure 9 encodes $\mathcal{R}$ in Example 25 together with on-demand E-strategies for* `s`*,* `:`*,* `2nd`*, and* `from` *[104]. Note that the second argument of '$:$' is evaluated* on-demand*. The evaluation of* $\mathsf{from}(\mathsf{s}(\mathsf{0}))$ *in* $\mathsf{2nd}(\mathsf{0} : \mathsf{from}(\mathsf{s}(\mathsf{0})))$ *is* demanded *by the rule defining* `2nd`*, and index -2 in $\xi(:)$ permits a reduction step. However, the third step of (27) is* not *demanded by the rule; thus, the infinite sequence is not possible. In contrast, the sequence (26) is possible with ξ, which obtains the normal form.*

The *on-demand evaluation* strategy (ODE) [5, 6] refines the on-demand E-strategy and has better computational properties; also, a transformation for proving termination of ODE as termination of *CSR* is given. Termination of `TEST` in Figure 9 can be proved in this way.

9.3. Lazy rewriting

In [83] the *lazy* graph rewriting of [43, 73] is formalized as term rewriting over *labeled* terms which carry information about the *reducibility* state of the positions in the term: *eager*, if they can be freely reduced, or *lazy* if they *block* reductions (on, and also below them) until some *activation* condition is raised.

Example 27. *As in [83, Example 3.1], consider $\mathcal{R}$ in Example 25 and $\mu(:) = \mu(\mathsf{2nd}) = \mu(\mathsf{from}) = \mu(\mathsf{s}) = \{1\}$. In lazy rewriting, the replacement map is used to* label *the terms. The* intended labelling *of $s = \mathsf{2nd}(\mathsf{0} : \mathsf{from}(\mathsf{s}(\mathsf{0})))$ is*

$$t = \mathit{label}_\mu(s) = \mathsf{2nd}^e(\mathsf{0}^e :^e \mathsf{from}^\ell(\mathsf{s}^e(\mathsf{0}^e)))$$

The intended labeling starts from the root of a term t, which is always labeled as *eager* (e); then, for each subterm $f(t_1, \ldots, t_k)$ of t, the root of each immediate subterm t_i is labeled with e or ℓ depending on whether $i \in \mu(f)$ or $i \notin \mu(f)$, respectively. Each lazy rewriting step on labelled terms may have two different effects:

1. changing the status (active or not) of a given position within a labelled term by means of an *activation* relation $\stackrel{\mathsf{A}}{\to}$ between labelled terms, or
2. performing a rewriting step on an active position by means of the relation of *active rewriting* $\stackrel{\mathsf{R}}{\to}_\mu$ between labelled terms.

Then, $\stackrel{\mathsf{LR}}{\to}_\mu = \stackrel{\mathsf{A}}{\to} \cup \stackrel{\mathsf{R}}{\to}_\mu$. A TRS is $LR(\mu)$-terminating if, for all $s \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, no infinite $\stackrel{\mathsf{LR}}{\to}_\mu$-rewrite sequence starts from $label_\mu(s)$. Schernhammer and Gramlich develop a transformation from a CS-TRS $(\mathcal{R}, \mu)$ (which defines lazy rewriting) into another CS-TRS $(\widetilde{\mathcal{R}}, \widetilde{\mu})$ which *characterizes* $LR(\mu)$-termination [122].

9.4. Forbidden patterns

In [56] *CSR* is extended by using *patterns* to identify (as instances by some substitution) subterms whose reduction is *forbidden*. *Forbidden patterns* are triples $\langle t, p, \lambda\rangle$, where t is a term, $p \in \mathcal{P}os(t)$, and $\lambda \in \{h, b, a\}$ specifies how the pattern forbids reductions with respect to position p: (i) **h**ere at p, (ii) strictly **b**elow p, or (iii) strictly **a**bove (but not at, below or parallel to) p [56, Definition 1]. Given a term s, a pattern $\pi = \langle t, p, \lambda\rangle$ determines a set $P_{t,p}(s) \subseteq \mathcal{P}os(s)$ of positions as follows: for all $q \in \mathcal{P}os(s)$, $q \in P_{t,p}(s) \Leftrightarrow s|_{q'} = \sigma(t) \wedge q = q'.p$ for some substitution σ and position q', i.e., $P_{t,p}(s)$ is the set of positions q of s which are obtained by *extending* with the component p of π (so that $q = q'.p$) the position q' of a subterm $s|_{q'}$ of s matched by the component t of π. Thus, $P_{t,p}$ defines a kind of *frontier* set of positions in s from which the parameter λ of π establishes whether we take positions in s which are *above* or *below* such frontier, or exactly *here*, in the frontier. Accordingly, $P_\pi(s)$ is as follows:

$$P_\pi(s) \quad = \quad \begin{cases} \{q' \in \mathcal{P}os(s) \mid \exists q \in P_{t,p}(s) \mid q' < q\} & \text{if } \lambda = a \\ \{q' \in \mathcal{P}os(s) \mid \exists q \in P_{t,p}(s) \mid q' > q\} & \text{if } \lambda = b \\ P_{t,p}(s) & \text{if } \lambda = h \end{cases}$$

Finally, given a term s and a set Π of forbidden patterns, $\overline{\mathcal{P}os}^\Pi(s) = \bigcup_{\pi\in\Pi} P_\pi(s)$ is the set of *forbidden positions* associated to s and $\mathcal{P}os^\Pi(t) = \mathcal{P}os(t) - \overline{\mathcal{P}os}^\Pi(t)$ is the set of *allowed* positions for rewriting: $s \to_\Pi t$ if $s \stackrel{p}{\to} t$ for some $p \in \mathcal{P}os^\Pi(s)$ [57, Section 2].

Example 28. *As in [56, Example 1], consider $\mathcal{R}$ in Example 25 and the set of forbidden patterns $\Pi = \{\langle x : (y : \mathsf{from}(z)), 2.2, h\rangle\}$ in [56, Example 2], containing a single forbidden pattern actually. Figure 10 depicts the positions of $s = \mathsf{2nd}(\mathsf{0} : \mathsf{s}(\mathsf{0}) : \mathsf{from}(\mathsf{s}(\mathsf{s}(\mathsf{0}))))$. For $\pi = \langle t, p, \lambda\rangle = \langle x : (y : \mathsf{from}(z)), 2.2, h\rangle$, and $q' = 1$, $s|_1 = \mathsf{0} : \mathsf{s}(\mathsf{0}) : \mathsf{from}(\mathsf{s}(\mathsf{s}(\mathsf{0})))$ is an instance of $\pi = x : (y : \mathsf{from}(z))$.*

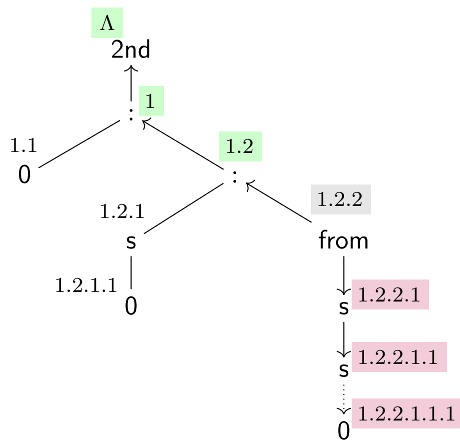


Figure 10: Positions of s above, below, or here w.r.t. $p = 2.2$ for π in Example 28

Thus, $q'.p = 1.2.2$ is in $P_{t,p}$. Actually, it is the only *one, i.e., $P_{t,p} = \{1.2.2\}$. Therefore, since $\lambda = h$ in π, $\overline{\mathcal{P}os}^{\Pi}(s) = \{1.2.2\}$. This means that (only) subterm* $\mathsf{from}(\mathsf{s}(\mathsf{s}(0)))$ *of s is not $\to_\Pi$-reducible in (27).*

Gramlich and Schernhammer show that $\to_\Pi$ is terminating and also head-normalizing for $\mathcal{R}$ [56, Examples 6 and 11].

Forbidden patterns can be used to specify restrictions of rewriting like *innermost* or *outermost* rewriting. Regarding *CSR*, $\hookrightarrow_\mu = \to_{\Pi_{csr}}$ for

$$\Pi_{csr} = \{\langle f(x_1,\ldots,x_k), i, h\rangle, \langle f(x_1,\ldots,x_k), i, b\rangle \mid f \in \mathcal{F}, i \in \{1,\ldots,k\} - \mu(f)\}$$

see [56, Section 3]. However, some of their combinations, like *innermost* CSR, can *not* be simulated by using forbidden patterns. The notion of *canonical forbidden pattern* [56, Definition 4] generalizes the canonical replacement maps to rewriting with forbidden patterns [56, Section 5]. Some methods for proving termination of rewriting with forbidden patterns are also given [57].

9.5. Controlled term rewriting

Controlled term rewriting [70] can be used to define *restrictions of the rewriting relation* by means of *selection automata* that *select positions in a term.* Controlled TRSs consist of rules of the form $\mathcal{A} : \ell \to r$, where $\mathcal{A}$ is a selection automaton and $\ell \to r$ is an ordinary rewrite rule. The rewriting steps on a term s with a rule $\mathcal{A} : \ell \to r$ are restricted to the positions p of redexes $\sigma(\ell)$ in s that are accepted by $\mathcal{A}$. As remarked by the authors, *"context-sensitive rewriting is a particular case of controlled rewriting"*. Actually, it is a strict subcase because *"the root position is always rewritable whereas this is not the case for controlled rewriting"* [70, page 181]. *Prefix-constrained* TRSs (pCTRSs) [71] are a proper subclass of controlled TRSs where the rewritable positions of a term are those whose *prefix* is accepted by a finite automaton. Given a CS-TRS $(\mathcal{R}, \mu)$, a pCTRS $\mathcal{P}$ generating $\hookrightarrow_{\mathcal{R},\mu}$ is obtained as follows [8, Section 3]: let $L = \Sigma^*$ for the alphabet $\Sigma = \{\langle f, i\rangle \mid f \in \mathcal{F}, i \in \mu(f)\}$. Then, $\mathcal{P} = \{L : \ell \to r \mid \ell \to r \in \mathcal{R}\}$.

Example 29. *For the TRS* $\mathcal{R}$

$$\begin{array}{rcl@{\qquad\qquad}rcl} \mathsf{a} & \to & \mathsf{b} & \mathsf{f}(\mathsf{h}(x,y),y) & \to & \mathsf{g}(x,y) \\ \mathsf{h}(\mathsf{a},\mathsf{b}) & \to & \mathsf{i}(\mathsf{a}) & & & \end{array}$$

with $\mu(\mathsf{i}) = \mu(\mathsf{g}) = \varnothing$ *and* $\mu(\mathsf{f}) = \mu(\mathsf{h}) = \{1\}$ *in [8, Example 26],* $\mathcal{P} = \{L : \ell \to r \mid \ell \to r \in \mathcal{R}\}$ *where* $L = \Sigma^*$ *for* $\Sigma = \{\langle \mathsf{f}, 1\rangle, \langle \mathsf{h}, 1\rangle\}$.

Controlled and prefix-constrained rewriting also fit the framework for replacement restrictions in Section 9.1 and can be seen as an interesting way to further develop it, using selection automata to describe the sets of active positions. However, controlled and prefix-constrained rewriting is not purely syntactic because each *rule* has an associated selection automaton whose behavior could be different for different rules of the same symbol. In this sense, it is more powerful than the approach in Section 9.1.

Prefix-constrained and controlled rewrite systems can be transformed into ordinary TRSs so that termination (tools and techniques) for TRSs can be used to prove and disprove termination of controlled rewriting [7]. This transformation extends and simplifies the (sound and complete) transformation in [48] for proving termination of CSR. Confluence of pCTRSs is investigated in [8] by extending the analysis of (local) confluence for *CSR* in [77].

10. Analysis of OBJ programs

In OBJ programs an operator *evaluation strategy* for a k-ary symbol f is a sequence $\xi(f) = (i_1\, i_2\, \cdots\, i_n)$, where $i_j \in \{0, \ldots, k\}$ for all $1 \leq j \leq n$. The evaluation of a term $t = f(t_1, \ldots, t_k)$ proceeds by considering the $i_1, \ldots, i_n$-th immediate subterms of t from left to right. If $i_j = 0$, then an attempt to apply a rule to t' is made (where t' is t with $t_{i_1}, \ldots, t_{i_{j-1}}$ replaced by their evaluated versions $t'_{i_1}, \ldots, t'_{i_{j-1}}$); otherwise, the evaluation of t_{i_j} (into t'_{i_j}) is recursively accomplished. The order of indices $i_1, i_2, \ldots, i_n$ determines the evaluation order of the immediate subterms of t. If no explicit evaluation strategy is given (i.e., all k-ary function symbols use the default strategy `(1 2` $\cdots$ `k 0)`), then Maude's evaluation strategy (in functional modules [19, Chapter 4]) corresponds to *leftmost-innermost evaluation*.

Example 30. *Consider the functional module in Figure 11, where sort* `NatList` *represents finite lists of natural numbers and* `NatIList` *represents possibly infinite lists. Symbol* `cons` *is* overloaded *and* `take` *can be used to obtain a given number of the initial components of a list. Besides illustrating the specification of E-strategies in* OBJ *programs, we will use it to illustrate the use (and limitations) of the current theory of* CSR *to analyze their properties.*

```
fmod InfListsAndTake is
   sorts Nat NatList NatIList .
   subsorts NatList < NatIList .
   op 0 : -> Nat .
   op s : Nat -> Nat .
   op nil : -> NatList .
   op cons : Nat NatIList -> NatIList [strat (0)].
   op cons : Nat NatList -> NatList [strat (0)].
   op inf : Nat -> NatIList .
   op take : Nat NatIList -> NatList  [strat (2 0)].
   vars M N : Nat .
   var  IL : NatIList .
   eq inf(N) = cons(N,inf(s(N))) .
   eq take(0, IL) = nil .
   eq take(s(M), cons(N, IL)) = cons(N, take(M, IL)) .
endfm
```

Figure 11: Example of Maude program

	Evaluation strategies (ξ)	Frozenness annotations (φ)
Available in	CafeOBJ, OBJ2, OBJ3, Maude	Maude
Use in Maude	Functional modules	System modules
Definition	$\xi(f)$ seq. of $i \in \{0,\ldots,ar(f)\}$	$\varphi(f) \subseteq \{1,\ldots,ar(f)\}$
Intended use	Defines the ev. strategy	Replacement restrictions
Rep. map	$\mu^\xi(f) = \xi(f) \cap \mathbb{N}_{>0}$	$\mu_\varphi(f) = \{1,\ldots,k\} - \varphi(f)$

Table 1: Use of context-sensitive rewriting in OBJ languages

Frozenness annotations in [17] were added later to Maude [19, Section 4.4.9] for use with *system* modules [19, Chapter 6].[20] As for frozenness annotations, a replacement map μ^ξ can be associated to ξ so that (i) the sequence becomes a set and (ii) indices 0 are removed: $\mu^\xi(f) = \xi(f) - \{0\}$, or more precisely

$$\mu^\xi(f) = \{i_1,\ldots,i_n \mid \xi(f) = (i_1\, i_2\, \cdots\, i_n)\} - \{0\}$$

Both uses of CS replacement restrictions in OBJ languages (in particular in Maude, as frozenness annotations are not available in CafeOBJ, OBJ2, or OBJ3) are summarized and compared in Table 1. We often represent computations with OBJ programs $\mathcal{R}$ which can be seen as TRSs and use evaluation strategies ξ or frozenness annotations φ by means of reduction relations $\to_{\mathcal{R},\xi}$ and $\to_{\mathcal{R},\varphi}$, respectively.

Remark 7 (*CSR* and OBJ computations). *Both $\to_{\mathcal{R},\xi}$ and $\to_{\mathcal{R},\varphi}$ perform*

[20]This is part of recent presentations of *Rewriting Logic* [94], which provides the theoretical basis for Maude, which "supports both forms of context-sensitive rewriting: with equations using the `strat` attribute, and with rules using the `frozen` attribute" [95, page 736].

μ^ξ*-reduction or* μ_φ*-reduction steps with* $\mathcal{R}$*, respectively. Actually,*

$\to_{\mathcal{R},\xi} \subseteq \hookrightarrow_{\mathcal{R},\mu^\xi}$ *for* CafeOBJ, OBJ* *programs, and* Maude *functional modules* $\mathcal{R}$
$\to_{\mathcal{R},\varphi} = \hookrightarrow_{\mathcal{R},\mu_\varphi}$ *for* Maude *system modules* $\mathcal{R}$

For evaluation strategies, $\to_{\mathcal{R},\xi} = \hookrightarrow_{\mathcal{R},\mu^\xi}$ *does not hold (in general) due to the* strategic *component of E-strategies which not only prevents the evaluation of some arguments in function calls, but also establishes an* order *for such an evaluation. In some cases a closer correspondence with* innermost CSR *can be obtained, see [80, 81]. Frozenness annotations* φ *provide a closer (although dual) correspondence:* φ *is a replacement map* μ_φ *where, for each symbol* f*, the set of frozen arguments in* $\varphi(f)$ *are* out *of* $\mu_\varphi(f)$ *and vice versa. Hence,* $\to_{\mathcal{R},\varphi} = \hookrightarrow_{\mathcal{R},\mu_\varphi}$.

Remark 7 summarizes the basis for the use of *CSR* in the analysis of OBJ programs which can be seen as TRSs, as we assume in the remainder of this section, unless stated otherwise. In the following, we discuss the connection between μ-normal forms and expressions computed by OBJ programs (Section 10.1). We explore the use of normalization via μ-normalization in Maude by using its *strategy language* (Section 10.2). The use of termination of *CSR* to prove termination of OBJ programs is discussed in Section 10.3. Then, we briefly discuss the use of *CSR* in the analysis of behavioral CafeOBJ specifications (Section 10.4); for modeling π-calculus in Maude (Section 10.5); and in Real-Time Maude (Section 10.6).

10.1. Computing μ*-normal forms in* OBJ *programs*

Since $\to_{\mathcal{R},\varphi} = \hookrightarrow_{\mathcal{R},\mu_\varphi}$, the evaluation of expressions in a TRS-like system module $\mathcal{R}$ with frozenness annotation φ returns μ_φ-normal forms. Hence, results and techniques using μ-normal forms in head-normalization, normalization and infinitary normalization can be used with Maude system modules (Section 10.2).

The evaluation of expressions with evaluation strategies ξ, though, returns *E-normal forms* (*ENF*s), i.e., terms which cannot be further rewritten using ξ. If for all *defined* symbols $f \in \mathcal{D}$, $\xi(f)$ ends in 0 (see [28, 102] for details), we have:

Theorem 17. [80] *Let* $\mathcal{R} = (\mathcal{C} \uplus \mathcal{D}, R)$ *be a TRS and* ξ *be an evaluation strategy such that for all* $f \in \mathcal{D}$*,* $\xi(f)$ *ends in* 0*. Every* ENF t *is a* μ^ξ*-normal form.*

Thus, under this reasonable condition, the aforementioned results and techniques for head-normalization, etc., are available for evaluation strategies as well.

10.2. Normalization via μ*-normalization in* Maude

For left-linear TRS-like OBJ programs $\mathcal{R}$ returning μ-normal forms (for $\mu = \mu_\varphi$ of $\mu = \mu^\xi$) for all initial expressions s, and such that $\mu \in CM_{\mathcal{R}}$, canonical *CSR* can be used to obtain head-normal forms, values, and (infinite)

```
smod NORM_VIA_MUNORM is
  protecting ExSec11_1_Luc02 .
  vars x y : S .

  strat norm_via_munorm @ S .
  strat munorm @ S .  strat decomp @ S .  strat dsuc @ S .  strat dcons @ S .
  strat drecip @ S .  strat dadd @ S .    strat ddbl @ S .  strat dhalf @ S .
  strat dsqr @ S .    strat dfirst @ S .  strat dterms @ S .

  sd norm_via_munorm := munorm ; try(decomp) .
  sd munorm := one(all) ! .
  sd decomp := dsuc | dcons | drecip | dadd | ddbl | dhalf
             | dsqr | dfirst | dterms .
  sd dsuc := matchrew s(x) by x using norm_via_munorm .
  sd dcons := matchrew (x : y) by x using norm_via_munorm , y
            using norm_via_munorm .
  sd drecip := matchrew recip(x) by x using norm_via_munorm .
  sd dadd := matchrew add(x,y) by x using norm_via_munorm , y
            using norm_via_munorm .
  sd ddbl := matchrew dbl(x) by x using norm_via_munorm .
  sd dhalf := matchrew half(x) by x using norm_via_munorm .
  sd dsqr := matchrew sqr(x) by x using norm_via_munorm .
  sd dfirst := matchrew first(x,y) by x using munorm , y using norm_via_munorm .
  sd dterms := matchrew terms(x) by x using norm_via_munorm .
endsm
```

Figure 12: Maude strategy for normalization-via-μ-normalization

normal forms [86, Section 9]. In particular, normal forms can be obtained by *normalization-via-μ-normalization* (see Figure 3). In Maude, the strategy language [121] can be used to implement $\mathtt{norm}_\mu$ as a Maude strategy defined by the strategy module NORM_VIA_MUNORM (Figure 12). The main strategy component

```
sd norm_via_munorm := munorm ; try(decomp) .
```

specifies that norm_via_munorm consists of a sequence of two steps:

1. the computation of the μ-normal form u of the initial expression s, as the repeated application of rules until no further steps can be issued

   ```
   sd munorm := one(all) ! .
   ```

   from which only one reduct is chosen, followed by
2. the (attempt of) *decomposition* of u to recursively apply norm_via_munorm to $s_1, \ldots, s_k$ if $u = f(s_1, \ldots, s_k)$[21] (if u is a constant or a variable nothing happens). Such a decomposition:

   ```
   sd decomp := dsuc | dcons | drecip | dadd | ddbl | dhalf
              | dsqr | dfirst | dterms .
   ```

[21] This description of $\mathtt{norm}_\mu$ does *not* use the decomposition $u = C[s_1, \ldots, s_n]$ of the computed μ-normal forms u as for the *maximal replacing context* $C[\,] = MRC^\mu(u)$ to jump into maximal frozen parts $s_1, \ldots, s_n$. The equivalent decomposition $u = f(s_1, \ldots, s_n)$ permits a simpler implementation which avoids the detection of the entire maximal replacing contexts.

is a disjunction of `matchrew` operators with patterns $f(x_1, \ldots, x_k)$ for each k-ary function symbol f (with $k > 0$) to extract the immediate subterms to feed `norm_via_munorm` again. Generically, each disjunctive component $\mathtt{d}f$ of `decomp` is defined as follows:

$$\begin{array}{l}\mathtt{sd\ d}f = \mathtt{matchrew}\ f(x_1,\ldots,x_k)\ \mathtt{by}\\ \qquad\qquad x_1\ \mathtt{using\ norm\_via\_munorm}, \ldots, x_k\ \mathtt{using\ norm\_via\_munorm}\end{array}$$

If $\mathcal{R}$ is left-linear, $\mu \in CM_{\mathcal{R}}$ and $\mathcal{R}$ is confluent and μ-terminating, then $\mathtt{norm}_\mu(s)$ obtains the normal form of any normalizing term s, see [86, Section 9]. For $\mathcal{R}$ in Example 1, we can use `norm_via_munorm` to obtain the first 4 components of the sequence approximating $\pi^2/6$, using command `dsrew`:

```
Maude> dsrew first(s(s(s(s(0)))),terms(s(0)))
    using norm_via_munorm .
    dsrewrite in NORM_VIA_MUNORM :
    first(s(s(s(s(0)))), terms(s(0))) using norm_via_munorm .

Solution 1
rewrites: 73 in 1ms cpu (2ms real) (45653 rewrites/second)
result S: recip(s(0)) : (recip(s(s(s(s(0)))))
  : (recip(s(s(s(s(s(s(s(s(s(0))))))))))
  : (recip(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(0))))))))))))))))))
  : nil)))

No more solutions.
rewrites: 73 in 1ms cpu (2ms real) (42690 rewrites/second)
```

where the obtained expression represents the expected list $[\frac{1}{1}, \frac{1}{4}, \frac{1}{9}, \frac{1}{16}]$. Using meta-level features of Maude 3.0 [22, Section 8], Rubén Rubio has developed a transformation which automatically constructs the `norm-via-munorm` strategy for any Maude functional or system module and permits a direct use of normalization-via-μ-normalization for a given initial expression, see

`http://maude.ucm.es/strategies/examples/munorm.maude`

10.3. Termination of OBJ *programs*

Since the reduction relation of TRS-like OBJ programs is included in the corresponding one-step μ-rewrite relation (see Remark 7), termination of (innermost) *CSR* provides a sufficient (and necessary, for Maude system modules) termination criterion [80, 81]; also [41, 42, 52]. In [26, 23, 88] a sequence of theory transformations is used to bridge the gap between termination of Maude programs and termination of *CSR*. In this setting, given a membership equational program $\mathcal{P}$, a CS-TRS $(\mathcal{R}_{\mathcal{P}}, \mu_{\mathcal{P}})$ is obtained whose termination implies that of $\mathcal{P}$.

Remark 8. *The replacement restrictions in $\mu_{\mathcal{P}}$ are (partly) due to the definition of the transformation* itself. *Thus, the replacement restrictions are actually part of the appropriate definition of the transformation.*

Further developments are reported in [25]. An important outcome of this research was the development of the *Maude Termination Tool* (MTT [24]). The tool transforms Maude programs into CS-TRSs and uses mu-term and AProVE as backends to obtain proofs of termination of the program. More information, examples of use, and benchmarks can be found here:

`http://www.lcc.uma.es/~duran/MTT`

Example 31. *The frozenness annotations in* `ExSec11_1_Luc02` *make it terminating. Also, the E-strategy in* `InfListsAndTake` *makes it terminating. Both claims can be automatically proved by using* MTT.

10.4. Analysis of (behavioral) CafeOBJ *specifications*

CafeOBJ behavioral specifications are modules that contain a special sort $\mathcal{H}$ (the *hidden* sort) and associated operation symbols called *behavioral* operation symbols [105, Section 4]. Sorts that are not hidden are called *visible*, and $\mathcal{V}$ is the set of visible sorts. Behavioral operation symbols are required to have exactly one hidden sort in their arguments, i.e., if $f : s_1 \times \cdots \times s_k \to s$ is behavioral (denoted $f \in \Sigma^b$), then one and only one of the s_i belongs to $\mathcal{H}$. The central concept in behavioral specification is *behavioral equivalence*, which is defined by the notion of *behavioral context*. A context $C[\ ]_p$ is said to be *behavioral* if all symbols in the path above position p are behavioral. The context is called *visible* if its sort is in $\mathcal{V}$. Then, given a sorted Σ-algebra $\mathcal{A} = (A, \Sigma_{\mathcal{A}})$, where A is an S-indexed set $A = \{A_s \mid s \in S\}$, two elements $a, a' \in A_s$ are behaviorally equivalent (written $a \sim a'$) if they are not distinguished by any behavioral operation symbol, i.e., $[\![C[a]]\!]_{\mathcal{A}} = [\![C[a']]\!]_{\mathcal{A}}$ for all *visible* behavioral contexts $C[\ ]$ [105, Definition 29]. Then, the authors call a non-behavioral symbol $f : s_1 \times \cdots \times s_k \to s$ to be *behaviorally coherent* for a model $\mathcal{A}$ of a given behavioral specification if for all $\vec{a}, \vec{b}$ with $a_i, b_i \in A_{s_i}$ for all $1 \leq i \leq k$, $a_i, b_i \in A_{s_i}$ are such that $a_i \sim b_i$, then $[\![f(\vec{a})]\!] \sim [\![f(\vec{b})]\!]$ [105, Definition 31]. [105, page 566] defines a replacement map μ^{BC} and proves that μ^{BC}-normalization and, in particular, μ^{BC}-termination of the behavioral specification can be used to check behavioral coherence of CafeOBJ specifications [105, Theorem 47] and [105, page 572].

10.5. Use of CSR *to model* π*-calculus in* Maude

Milner's π-calculus [97, 98, 99] is a computational scheme which models concurrency. The set $\mathcal{P}$ of processes $P \in \mathcal{P}$ is defined by: $P ::= \Sigma_{i\in I}\pi_i.P_i \mid (P \mid Q) \mid\ !P \mid (\nu x)P$, where $\pi_i ::= x(y) \mid \overline{x}y$, for $x, \overline{x}, y \in \mathcal{X}$ (a set of *names*), represents the basic communication actions: *input* ($x(y)$) and *output* ($\overline{x}y$)); and the process constructors are '|' (parallelism), '!' (replication), νx (restriction) and '+' (nondeterministic choice). Some expressions are identified by means of a congruence $\equiv$ on $\mathcal{P}$. The *transition relation* $\to\ \subseteq \mathcal{P} \times \mathcal{P}$ that formalizes the reduction process is defined by the axiom (COMM) and the rules below [97]:

$$\text{COMM:}(\cdots + x(y).P) \mid (\cdots + \overline{x}z.Q) \to P[z/y] \mid Q$$

$$\text{PAR:} \quad \frac{P \to P'}{P \mid Q \to P' \mid Q} \qquad \text{RES:} \quad \frac{P \to P'}{(\nu x)P \to (\nu x)P'}$$

$$\text{STRUCT:} \quad \frac{Q \equiv P, P \to P', P' \equiv Q'}{Q \to Q'}$$

Note the absence of context-passing rules for replication (!) and choice (+). Thatti, Sen, and Martí-Oliet describe an executable specification of the operational semantics of an asynchronous version of the π-calculus in Maude [127]. In their specification, *each of the non-constant syntax constructors is declared as frozen, so that the corresponding arguments cannot be rewritten by rules* [127, page 264]. This is necessary, not only to faithfully represent the operational semantics of the calculus (see [127, Table 2]) but also to avoid the ill-formed terms, see [127, Section 3].

10.6. Use of eager and lazy rewrite rules in Real-Time Maude

Ölveczky and Meseguer have shown how to use replacement restrictions to implement *eager* and *lazy* rewrite rules in real-time and hybrid systems in rewriting logic [118]. Eager and lazy rewrite rules were introduced in [117] *to model* urgency *by letting the application of eager rules take precedence over the application of lazy tick rules* that model the elapse of time on a system [117, Section 2.2]. In [118], taking benefit from replacement restrictions in Maude (see Section 3), a new encoding of *eager* and *lazy* rules is provided so that there is no need to treat them asymmetrically, in an ad-hoc manner, in the implementation of Real-Time Maude 2.1 [118, Section 3.2].

11. Conclusions

We have given an overview on applications and extensions of *CSR* reported by several authors during the last 20 years. Such applications and extensions come from quite different subfields of term rewriting and rewriting-based programming languages: termination analysis and strategies, conditional rewriting, productivity, computational complexity, etc. We made an effort to use a uniform notation for material coming from different authors, and also to introduce unifying approaches hopefully helpful to draw connections among apparently disconnected fields. We clarify some insufficiently discussed aspects of the considered applications and extensions.

We also provide some new results (marked with $^{(\star)}$), in particular, Proposition 1 shows that proving $\mu^{\mathcal{U}}$-termination of $\mathcal{U}(\mathcal{R})$ improves on just proving termination of $\mathcal{U}(\mathcal{R})$ when trying to prove operational termination of CTRSs $\mathcal{R}$. Theorem 8 provides a new criterion of (non-)confluence for CTRSs by using transformation $\mathcal{U}$ while Theorem 15 provides a new criterion of confluence for CS-CTRSs by using transformation $\mathcal{U}_{opt}$. We plan to implement them in the near future. Also, examples are given to illustrate the use or relevance of the different applications or results gathered in the paper. Many of them come from the literature, although in most cases we extended their scope in some way. For instance, the program in Figure 1 is an excerpt of a TRS in [82, Section 11.1], but its use as a Maude system module is new (in [86] the same TRS was handled as a Maude functional module). Other examples are new (e.g., Examples 5, 14, 15, 16, 21, 28, and 32). Besides, Remark 5 stresses how Hirokawa and Moser provided, as a 'side effect' of their work on runtime complexity of TRSs, the first results regarding runtime complexity (bounds) for *CSR*.

There also are new contributions to the use of *CSR* in rewriting-based languages like Maude, with the development of an implementation of normalization-via-μ-normalization using Maude's strategy language (see Section 10.2), and also showing the use of replacement maps in Maude system modules, both for rewriting (Section 3) and narrowing (Section 8.4).

11.1. Future work

Regarding possible avenues of further research and cross-fertilization, several paragraphs in the development point to underexplored aspects deserving further research (e.g., the analysis of operational termination of CS-CTRSs using dependency pairs, the analysis of confluence of CS-CTRSs, termination of AC-CS-TRSs using dependency pairs, etc.). Also, Section 10 discusses a number of applications of the theory of *CSR* in the analysis of sophisticated programming languages in use like CafeOBJ and Maude. Programs in such languages are more sophisticated than TRSs and many of their features may concern correctness and completeness of computations in ways not sufficiently covered by the current theory of *CSR*, which essentially focuses on TRSs. For instance, in Section 8.1 we discuss the mismatch between the canonical replacement map and the use of conditional rules, that leads to 'bad' properties of *ENF*s like not being head-normal forms. A partial solution has been provided for normal CTRSs, but how to obtain normal forms in this setting? Is there a normalization-via-μ-normalization process that applies? These are subjects deserving further research. Also, *sort* information could be used to *improve* the definition of canonical replacement maps in sorted TRSs and Maude programs to guarantee good computational properties.

Example 32. *Consider the Maude functional module in Figure 11. Although the E-strategy ξ forbids reductions on the first argument of* `take` *and $\mu^\xi \notin CM_{\mathcal{R}}$, calls to* `take` *can be* completely *evaluated. This is because* there is no equation associated to any function of sort `Nat`*. Thus, reductions on the first argument of* `take` *are actually* impossible *(rather than forbidden!). Also, the program is* completely defined *due to the sort discipline.*

Again, these issues deserve further investigation. Since modern rewriting-based programming languages like CafeOBJ and Maude often combine these features (and more), this brief discussion shows that more research is necessary to provide more appropriate support of context-sensitivity in such computational settings.

Acknowledgements

I thank Santiago Escobar for his help in the use of narrowing in Maude. I also thank Narciso Martí-Oliet and Rubén Rubio for their help in the details of the Maude strategy language. Thanks are also due to Nao Hirokawa and Georg Moser for their comments on Remark 5. Finally, I'm grateful to Jürgen Giesl, José Meseguer, Masahiko Sakai, and Hans Zantema for reading and commenting on an earlier version of this manuscript.

References

[1] Alarcón, B., Emmes, F., Fuhs, C., Giesl, J., Gutiérrez, R., Lucas, S., Schneider-Kamp, P., Thiemann, R., 2008. Improving Context-Sensitive Dependency Pairs. In: Cervesato, I., Veith, H., Voronkov, A. (Eds.), Logic for Programming, Artificial Intelligence, and Reasoning, 15th International Conference, LPAR 2008, Doha, Qatar, November 22-27, 2008. Proceedings. Vol. 5330 of Lecture Notes in Computer Science. Springer, pp. 636–651.

[2] Alarcón, B., Gutiérrez, R., Lucas, S., 2010. Context-sensitive dependency pairs. Inf. Comput. 208 (8), 922–968.

[3] Alarcón, B., Lucas, S., 2009. Using Context-Sensitive Rewriting for Proving Innermost Termination of Rewriting. Electr. Notes Theor. Comput. Sci. 248, 3–17.

[4] Alarcón, B., Lucas, S., Meseguer, J., 2010. A Dependency Pair Framework for A OR C-Termination. In: Ölveczky, P. C. (Ed.), Rewriting Logic and Its Applications - 8th International Workshop, WRLA 2010, Held as a Satellite Event of ETAPS 2010, Paphos, Cyprus, March 20-21, 2010, Revised Selected Papers. Vol. 6381 of Lecture Notes in Computer Science. Springer, pp. 35–51.

[5] Alpuente, M., Escobar, S., Gramlich, B., Lucas, S., 2002. Improving On-Demand Strategy Annotations. In: Baaz, M., Voronkov, A. (Eds.), LPAR. Vol. 2514 of Lecture Notes in Computer Science. Springer, pp. 1–18.

[6] Alpuente, M., Escobar, S., Gramlich, B., Lucas, S., 2010. On-demand strategy annotations revisited: An improved on-demand evaluation strategy. Theor. Comput. Sci. 411 (2), 504–541.

[7] Andrianarivelo, N., Pelletier, V., Réty, P., 2017. Transforming Prefix-constrained or Controlled Rewrite Systems. In: Mosbah, M., Rusinowitch, M. (Eds.), SCSS 2017, The 8th International Symposium on Symbolic Computation in Software Science 2017, April 6-9, 2017, Gammarth, Tunisia. Vol. 45 of EPiC Series in Computing. EasyChair, pp. 49–62.

[8] Andrianarivelo, N., Réty, P., 2018. Confluence of Prefix-Constrained Rewrite Systems. In: Kirchner, H. (Ed.), 3rd International Conference on Formal Structures for Computation and Deduction, FSCD 2018, July 9-12, 2018, Oxford, UK. Vol. 108 of LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 6:1–6:15.

[9] Antoy, S., 1992. Definitional Trees. In: Kirchner, H., Levi, G. (Eds.), Algebraic and Logic Programming, Third International Conference, Volterra, Italy, September 2-4, 1992, Proceedings. Vol. 632 of Lecture Notes in Computer Science. Springer, pp. 143–157.

[10] Antoy, S., Hanus, M., 2010. Functional logic programming. Commun. ACM 53 (4), 74–85.

[11] Arts, T., Giesl, J., 2000. Termination of term rewriting using dependency pairs. Theor. Comput. Sci. 236 (1-2), 133–178.

[12] Baader, F., Nipkow, T., 1998. Term rewriting and all that. Cambridge University Press.

[13] Bergstra, J. A., Klop, J. W., 1986. Conditional rewrite rules: Confluence and termination. J. Comput. Syst. Sci. 32 (3), 323–362.

[14] Bonfante, G., Cichon, A., Marion, J., Touzet, H., 1998. Complexity Classes and Rewrite Systems with Polynomial Interpretation. In: Gottlob, G., Grandjean, E., Seyr, K. (Eds.), Computer Science Logic, 12th International Workshop, CSL '98, Annual Conference of the EACSL, Brno, Czech Republic, August 24-28, 1998, Proceedings. Vol. 1584 of Lecture Notes in Computer Science. Springer, pp. 372–384.

[15] Bonfante, G., Cichon, A., Marion, J., Touzet, H., 2001. Algorithms with polynomial interpretation termination proof. J. Funct. Program. 11 (1), 33–53.

[16] Bruni, R., Meseguer, J., 2003. Generalized Rewrite Theories. In: Baeten, J. C. M., Lenstra, J. K., Parrow, J., Woeginger, G. J. (Eds.), Automata, Languages and Programming, 30th International Colloquium, ICALP 2003, Eindhoven, The Netherlands, June 30 - July 4, 2003. Proceedings. Vol. 2719 of Lecture Notes in Computer Science. Springer, pp. 252–266.

[17] Bruni, R., Meseguer, J., 2006. Semantic foundations for generalized rewrite theories. Theor. Comput. Sci. 360 (1-3), 386–414.

[18] Cichon, A., Lescanne, P., 1992. Polynomial Interpretations and the Complexity of Algorithms. In: Kapur, D. (Ed.), Automated Deduction - CADE-11, 11th International Conference on Automated Deduction, Saratoga Springs, NY, USA, June 15-18, 1992, Proceedings. Vol. 607 of Lecture Notes in Computer Science. Springer, pp. 139–147.

[19] Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C. L., 2007. All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic. Vol. 4350 of Lecture Notes in Computer Science. Springer.

[20] Courcelle, B., 1983. Fundamental Properties of Infinite Trees. Theor. Comput. Sci. 25, 95–169.

[21] Dershowitz, N., Okada, M., Sivakumar, G., 1988. Canonical Conditional Rewrite Systems. In: Lusk, E. L., Overbeek, R. A. (Eds.), 9th International Conference on Automated Deduction, Argonne, Illinois, USA, May 23-26, 1988, Proceedings. Vol. 310 of Lecture Notes in Computer Science. Springer, pp. 538–549.

[22] Durán, F., Eker, S., Escobar, S., Martí-Oliet, N., Meseguer, J., Rubio, R., Talcott, C. L., 2020. Programming and symbolic computation in Maude. J. Log. Algebr. Meth. Program. 110.

[23] Durán, F., Lucas, S., Marché, C., Meseguer, J., Urbain, X., 2008. Proving operational termination of membership equational programs. High. Order Symb. Comput. 21 (1-2), 59–88.

[24] Durán, F., Lucas, S., Meseguer, J., 2008. MTT: The Maude Termination Tool (System Description). In: Armando, A., Baumgartner, P., Dowek, G. (Eds.), Automated Reasoning, 4th International Joint Conference, IJCAR 2008, Sydney, Australia, August 12-15, 2008, Proceedings. Vol. 5195 of Lecture Notes in Computer Science. Springer, pp. 313–319.

[25] Durán, F., Lucas, S., Meseguer, J., 2009. Methods for Proving Termination of Rewriting-based Programming Languages by Transformation. Electr. Notes Theor. Comput. Sci. 248, 93–113.

[26] Durán, F., Lucas, S., Meseguer, J., Marché, C., Urbain, X., 2004. Proving termination of membership equational programs. In: Heintze, N., Sestoft, P. (Eds.), Proceedings of the 2004 ACM SIGPLAN Workshop on Partial Evaluation and Semantics-based Program Manipulation, 2004, Verona, Italy, August 24-25, 2004. ACM, pp. 147–158.

[27] Durán, F., Meseguer, J., 2012. On the Church-Rosser and coherence properties of conditional order-sorted rewrite theories. J. Log. Algebraic Methods Program. 81 (7-8), 816–850.

[28] Eker, S., 1998. Term rewriting with operator evaluation strategies. In: Kirchner, C., Kirchner, H. (Eds.), 1998 International Workshop on Rewriting Logic and its Applications, WRLA 1998, Abbaye des Prémontrés at Pont-à-Mousson, France, September 1998. Vol. 15 of Electronic Notes in Theoretical Computer Science. Elsevier, pp. 311–330.

[29] Endrullis, J., Grabmayer, C., Hendriks, D., 2008. Data-Oblivious Stream Productivity. In: Cervesato, I., Veith, H., Voronkov, A. (Eds.), Logic for Programming, Artificial Intelligence, and Reasoning, 15th International Conference, LPAR 2008, Doha, Qatar, November 22-27, 2008. Proceedings. Vol. 5330 of Lecture Notes in Computer Science. Springer, pp. 79–96.

[30] Endrullis, J., Grabmayer, C., Hendriks, D., Isihara, A., Klop, J. W., 2007. Productivity of Stream Definitions. In: Csuhaj-Varjú, E., Ésik, Z. (Eds.), Fundamentals of Computation Theory, 16th International Symposium,

FCT 2007, Budapest, Hungary, August 27-30, 2007, Proceedings. Vol. 4639 of Lecture Notes in Computer Science. Springer, pp. 274–287.

[31] Endrullis, J., Grabmayer, C., Hendriks, D., Isihara, A., Klop, J. W., 2010. Productivity of stream definitions. Theor. Comput. Sci. 411 (4-5), 765–782.

[32] Endrullis, J., Hendriks, D., 2009. From Outermost to Context-Sensitive Rewriting. In: Treinen, R. (Ed.), Rewriting Techniques and Applications, 20th International Conference, RTA 2009, Brasília, Brazil, June 29 - July 1, 2009, Proceedings. Vol. 5595 of Lecture Notes in Computer Science. Springer, pp. 305–319.

[33] Endrullis, J., Hendriks, D., 2010. Transforming Outermost into Context-Sensitive Rewriting. Logical Methods in Computer Science 6 (2).

[34] Endrullis, J., Hendriks, D., 2011. Lazy productivity via termination. Theor. Comput. Sci. 412 (28), 3203–3225.

[35] Endrullis, J., Waldmann, J., Zantema, H., 2008. Matrix Interpretations for Proving Termination of Term Rewriting. J. Autom. Reasoning 40 (2-3), 195–220.

[36] Falke, S., December 2009. Term Rewriting with Built-In Numbers and Collection Data Structures. Ph.D. thesis, The University of New Mexico, Albuquerque, New Mexico.

[37] Falke, S., Kapur, D., 2008. Dependency Pairs for Rewriting with Built-In Numbers and Semantic Data Structures. In: Voronkov, A. (Ed.), Rewriting Techniques and Applications, 19th International Conference, RTA 2008, Hagenberg, Austria, July 15-17, 2008, Proceedings. Vol. 5117 of Lecture Notes in Computer Science. Springer, pp. 94–109.

[38] Falke, S., Kapur, D., 2009. Termination of Context-Sensitive Rewriting with Built-In Numbers and Collection Data Structures. In: Escobar, S. (Ed.), Functional and Constraint Logic Programming, 18th International Workshop, WFLP 2009, Brasilia, Brazil, June 28, 2009, Revised Selected Papers. Vol. 5979 of Lecture Notes in Computer Science. Springer, pp. 44–61.

[39] Fernández, M., 2005. Relaxing monotonicity for innermost termination. Inf. Process. Lett. 93 (3), 117–123.

[40] Ferreira, M. C. F., Ribeiro, A. L., 1999. Context-Sensitive AC-Rewriting. In: Narendran, P., Rusinowitch, M. (Eds.), Rewriting Techniques and Applications, 10th International Conference, RTA-99, Trento, Italy, July 2-4, 1999, Proceedings. Vol. 1631 of Lecture Notes in Computer Science. Springer, pp. 286–300.

[41] Fissore, O., Gnaedig, I., Kirchner, H., 2001. Induction for Termination with Local Strategies. Electr. Notes Theor. Comput. Sci. 58 (2), 155–188.

[42] Fissore, O., Gnaedig, I., Kirchner, H., 2003. Simplification and termination of strategies in rule-based languages. In: Proceedings of the 5th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, 27-29 August 2003, Uppsala, Sweden. ACM, pp. 124–135.

[43] Fokkink, W., Kamperman, J., Walters, P., 2000. Lazy rewriting on eager machinery. ACM Trans. Program. Lang. Syst. 22 (1), 45–86.

[44] Friedman, D. P., Wise, D. S., 1976. CONS Should Not Evaluate its Arguments. In: Michaelson, S., Milner, R. (Eds.), Third International Colloquium on Automata, Languages and Programming, University of Edinburgh, UK, July 20-23, 1976. Edinburgh University Press, pp. 257–284.

[45] Fuhs, C., 2019. Transforming Derivational Complexity of Term Rewriting to Runtime Complexity. In: Herzig, A., Popescu, A. (Eds.), Frontiers of Combining Systems - 12th International Symposium, FroCoS 2019, London, UK, September 4-6, 2019, Proceedings. Vol. 11715 of Lecture Notes in Computer Science. Springer, pp. 348–364.

[46] Giesl, J., Arts, T., 2001. Verification of Erlang Processes by Dependency Pairs. Appl. Algebra Eng. Commun. Comput. 12 (1/2), 39–72.

[47] Giesl, J., Aschermann, C., Brockschmidt, M., Emmes, F., Frohn, F., Fuhs, C., Hensel, J., Otto, C., Plücker, M., Schneider-Kamp, P., Ströder, T., Swiderski, S., Thiemann, R., 2017. Analyzing program termination and complexity automatically with aprove. J. Autom. Reasoning 58 (1), 3–31.

[48] Giesl, J., Middeldorp, A., 1999. Transforming Context-Sensitive Rewrite Systems. In: Narendran, P., Rusinowitch, M. (Eds.), Rewriting Techniques and Applications, 10th International Conference, RTA-99, Trento, Italy, July 2-4, 1999, Proceedings. Vol. 1631 of Lecture Notes in Computer Science. Springer, pp. 271–287.

[49] Giesl, J., Middeldorp, A., 2004. Transformation techniques for context-sensitive rewrite systems. J. Funct. Program. 14 (4), 379–427.

[50] Giesl, J., Schneider-Kamp, P., Thiemann, R., 2006. Automatic Termination Proofs in the Dependency Pair Framework. In: Furbach, U., Shankar, N. (Eds.), Automated Reasoning, Third International Joint Conference, IJCAR 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings. Vol. 4130 of Lecture Notes in Computer Science. Springer, pp. 281–286.

[51] Giesl, J., Thiemann, R., Schneider-Kamp, P., Falke, S., 2006. Mechanizing and Improving Dependency Pairs. J. Autom. Reasoning 37 (3), 155–203.

[52] Gnaedig, I., Kirchner, H., 2009. Termination of rewriting under strategies. ACM Trans. Comput. Log. 10 (2).

[53] Goguen, J. A., Meseguer, J., 1992. Order-Sorted Algebra I: Equational Deduction for Multiple Inheritance, Overloading, Exceptions and Partial Operations. Theor. Comput. Sci. 105 (2), 217–273.

[54] Goguen, J. A., Winkler, T., Meseguer, J., Futatsugi, K., Jouannaud, J.-P., 2000. Introducing OBJ Software Engineering with OBJ: algebraic specification in action, 169–236.

[55] Gramlich, B., 1995. Abstract Relations between Restricted Termination and Confluence Properties of Rewrite Systems. Fundam. Inform. 24 (1/2), 2–23.

[56] Gramlich, B., Schernhammer, F., 2009. Extending Context-Sensitivity in Term Rewriting. In: Fernández, M. (Ed.), 9th International Workshop on Reduction Strategies in Rewriting and Programming, WRS 2009. pp. 56–68.

[57] Gramlich, B., Schernhammer, F., 2010. Termination of Rewriting with and Automated Synthesis of Forbidden Patterns. In: Kirchner, H., Muñoz, C. (Eds.), Strategies in Rewriting, Proving, and Programming, IWS 2010. Vol. 44 of EPTCS. pp. 35–50.

[58] Gutiérrez, R., Lucas, S., 2010. Proving Termination in the Context-Sensitive Dependency Pair Framework. In: Ölveczky, P. C. (Ed.), Rewriting Logic and Its Applications - 8th International Workshop, WRLA 2010, Held as a Satellite Event of ETAPS 2010, Paphos, Cyprus, March 20-21, 2010, Revised Selected Papers. Vol. 6381 of Lecture Notes in Computer Science. Springer, pp. 18–34.

[59] Gutiérrez, R., Lucas, S., 2020. Automatically Proving and Disproving Feasibility Conditions. In: Peltier, N., Sofronie-Stokkermans, V. (Eds.), Automated Reasoning - 10th International Joint Conference, IJCAR 2020, Paris, France, July 1-4, 2020, Proceedings, Part II. Vol. 12167 of Lecture Notes in Computer Science. Springer, pp. 416–435.

[60] Gutiérrez, R., Lucas, S., 2020. mu-term: Verify Termination Properties Automatically (System Description). In: Peltier, N., Sofronie-Stokkermans, V. (Eds.), Automated Reasoning - 10th International Joint Conference, IJCAR 2020, Paris, France, July 1-4, 2020, Proceedings, Part II. Vol. 12167 of Lecture Notes in Computer Science. Springer, pp. 436–447.

[61] Hanus, M., 1994. The Integration of Functions into Logic Programming: From Theory to Practice. J. Log. Program. 19/20, 583–628.

[62] Hanus, M., 1995. Functional Logic Languages: Combine Search and Efficient Evaluation (Panel Abstract). In: Lloyd, J. W. (Ed.), ILPS. MIT Press, pp. 625–626.

[63] Hendrix, J., Meseguer, J., 2007. On the completeness of context-sensitive order-sorted specifications. In: Baader, F. (Ed.), RTA. Vol. 4533 of Lecture Notes in Computer Science. Springer, pp. 229–245.

[64] Hirokawa, N., Moser, G., 2008. Automated Complexity Analysis Based on the Dependency Pair Method. In: Armando, A., Baumgartner, P., Dowek, G. (Eds.), Automated Reasoning, 4th International Joint Conference, IJCAR 2008, Sydney, Australia, August 12-15, 2008, Proceedings. Vol. 5195 of Lecture Notes in Computer Science. Springer, pp. 364–379.

[65] Hirokawa, N., Moser, G., 2014. Automated Complexity Analysis Based on Context-Sensitive Rewriting. In: Dowek, G. (Ed.), Rewriting and Typed Lambda Calculi - Joint International Conference, RTA-TLCA 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings. Vol. 8560 of Lecture Notes in Computer Science. Springer, pp. 257–271.

[66] Hirokawa, N., Nagele, J., Middeldorp, A., 2018. Cops and CoCoWeb: Infrastructure for Confluence Tools. In: Galmiche, D., Schulz, S., Sebastiani, R. (Eds.), Automated Reasoning - 9th International Joint Conference, IJCAR 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings. Vol. 10900 of Lecture Notes in Computer Science. Springer, pp. 346–353.

[67] Hofbauer, D., 1992. Termination Proofs by Multiset Path Orderings Imply Primitive Recursive Derivation Lengths. Theor. Comput. Sci. 105 (1), 129–140.

[68] Hofbauer, D., Lautemann, C., 1989. Termination Proofs and the Length of Derivations (Preliminary Version). In: Dershowitz, N. (Ed.), Rewriting Techniques and Applications, 3rd International Conference, RTA-89, Chapel Hill, North Carolina, USA, April 3-5, 1989, Proceedings. Vol. 355 of Lecture Notes in Computer Science. Springer, pp. 167–177.

[69] Hudak, P., Peyton Jones, S. L., Wadler, P., Boutel, B., Fairbairn, J., Fasel, J. H., Guzmán, M. M., Hammond, K., Hughes, J., Johnsson, T., Kieburtz, R. B., Nikhil, R. S., Partain, W., Peterson, J., 1992. Report on the Programming Language Haskell, A Non-strict, Purely Functional Language. SIGPLAN Notices 27 (5), 1.

[70] Jacquemard, F., Kojima, Y., Sakai, M., 2011. Controlled Term Rewriting. In: Tinelli, C., Sofronie-Stokkermans, V. (Eds.), Frontiers of Combining Systems, 8th International Symposium, FroCoS 2011, Saarbrücken, Germany, October 5-7, 2011. Proceedings. Vol. 6989 of Lecture Notes in Computer Science. Springer, pp. 179–194.

[71] Jacquemard, F., Kojima, Y., Sakai, M., 2015. Term Rewriting with Prefix Context Constraints and Bottom-Up Strategies. In: Felty, A. P., Middeldorp, A. (Eds.), Automated Deduction - CADE-25 - 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings. Vol. 9195 of Lecture Notes in Computer Science. Springer, pp. 137–151.

[72] Jouannaud, J., Waldmann, B., 1987. Reductive conditional term rewriting systems. In: Wirsing, M. (Ed.), Formal Description of Programming Concepts - III: Proceedings of the IFIP TC 2/WG 2.2 Working Conference on Formal Description of Programming Concepts - III, Ebberup, Denmark, 25-28 August 1986. North-Holland, pp. 223–244.

[73] Kamperman, J. F. T., Walters, H. R., 1995. Lazy Rewriting and Eager Machinery. In: Hsiang, J. (Ed.), Rewriting Techniques and Applications, 6th International Conference, RTA-95, Kaiserslautern, Germany, April 5-7, 1995, Proceedings. Vol. 914 of Lecture Notes in Computer Science. Springer, pp. 147–162.

[74] Kaplan, S., 1987. Simplifying Conditional Term Rewriting Systems: Unification, Termination and Confluence. J. Symb. Comput. 4 (3), 295–334.

[75] Kapur, D., Narendran, P., Zhang, H., 1987. On Sufficient-Completeness and Related Properties of Term Rewriting Systems. Acta Informatica 24 (4), 395–415.

[76] Kop, C., Middeldorp, A., Sternagel, T., 2017. Complexity of Conditional Term Rewriting. Logical Methods in Computer Science 13 (1).

[77] Lucas, S., 1996. Context-sensitive computations in confluent programs. In: Kuchen, H., Swierstra, S. D. (Eds.), PLILP. Vol. 1140 of Lecture Notes in Computer Science. Springer, pp. 408–422.

[78] Lucas, S., 1997. Computational Properties of Term Rewriting with Replacement Restrictions. In: Falaschi, M., Navarro, M., Policriti, A. (Eds.), 1997 Joint Conf. on Declarative Programming, APPIA-GULP-PRODE'97, Grado, Italy, June 16-19, 1997. pp. 393–404.

[79] Lucas, S., 1998. Context-sensitive Computations in Functional and Functional Logic Programs. J. Funct. Log. Program. 1998 (1).

[80] Lucas, S., 2001. Termination of On-Demand Rewriting and Termination of OBJ Programs. In: Proceedings of the 3rd international ACM SIGPLAN conference on Principles and practice of declarative programming, September 5-7, 2001, Florence, Italy. ACM, pp. 82–93.

[81] Lucas, S., 2001. Termination of Rewriting With Strategy Annotations. In: Nieuwenhuis, R., Voronkov, A. (Eds.), Logic for Programming, Artificial Intelligence, and Reasoning, 8th International Conference, LPAR 2001,

Havana, Cuba, December 3-7, 2001, Proceedings. Vol. 2250 of Lecture Notes in Computer Science. Springer, pp. 669–684.

[82] Lucas, S., 2002. Context-Sensitive Rewriting Strategies. Inf. Comput. 178 (1), 294–343.

[83] Lucas, S., 2002. Lazy Rewriting and Context-Sensitive Rewriting. Electr. Notes Theor. Comput. Sci. 64, 234–254.

[84] Lucas, S., 2005. A note on completeness of conditional context-sensitive rewriting. In: 5th International Workshop on Reduction Strategies in Rewriting and Programming, WRS. University of Kyoto, pp. 3–15.

[85] Lucas, S., 2015. Termination of canonical context-sensitive rewriting and productivity of rewrite systems. In: Navarro, M. (Ed.), Proceedings XV Jornadas sobre Programación y Lenguajes, PROLE 2015, Santander, Spain, 15-17th September 2015. Vol. 200 of EPTCS. pp. 18–31.

[86] Lucas, S., 2020. Context-sensitive Rewriting. ACM Comput. Surv. 53 (4), 78:1–78:36.

[87] Lucas, S., Marché, C., Meseguer, J., 2005. Operational termination of conditional term rewriting systems. Inf. Process. Lett. 95 (4), 446–453.

[88] Lucas, S., Meseguer, J., 2009. Operational Termination of Membership Equational Programs: the Order-Sorted Way. Electr. Notes Theor. Comput. Sci. 238 (3), 207–225.

[89] Lucas, S., Meseguer, J., 2016. Normal forms and normal theories in conditional rewriting. J. Log. Algebr. Meth. Program. 85 (1), 67–97.

[90] Lucas, S., Meseguer, J., 2017. Dependency pairs for proving termination properties of conditional term rewriting systems. J. Log. Algebr. Meth. Program. 86 (1), 236–268.

[91] Lucas, S., Meseguer, J., Gutiérrez, R., 2018. The 2D Dependency Pair Framework for conditional rewrite systems. Part I: Definition and basic processors. J. Comput. Syst. Sci. 96, 74–106.

[92] Lucas, S., Meseguer, J., Gutiérrez, R., 2020. The 2D Dependency Pair Framework for Conditional Rewrite Systems. Part II: Advanced Processors and Implementation Techniques. J. Autom. Reasoning 64, 1611–1662.

[93] Marchiori, M., 1996. Unravelings and Ultra-properties. In: Hanus, M., Rodríguez-Artalejo, M. (Eds.), Algebraic and Logic Programming, 5th International Conference, ALP'96, Aachen, Germany, September 25-27, 1996, Proceedings. Vol. 1139 of Lecture Notes in Computer Science. Springer, pp. 107–121.

[94] Meseguer, J., 1992. Conditional Rewriting Logic as a Unified Model of Concurrency. Theor. Comput. Sci. 96 (1), 73–155.

[95] Meseguer, J., 2012. Twenty years of rewriting logic. J. Log. Algebr. Program. 81 (7-8), 721–781.

[96] Middeldorp, A., 1997. Call by Need Computations to Root-Stable Form. In: Lee, P., Henglein, F., Jones, N. D. (Eds.), Conference Record of POPL'97: The 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Papers Presented at the Symposium, Paris, France, 15-17 January 1997. ACM Press, pp. 94–105.

[97] Milner, R., 1991. The polyadic π-calculus: A tutorial. In: F.L. Brauer, W. B., Schwichtenberg, H. (Eds.), Logic and Algebra of Specifications. Springer-Verlag.

[98] Milner, R., 1999. Communicating and mobile systems - the Pi-calculus. Cambridge University Press.

[99] Milner, R., Parrow, J., Walker, D., 1992. A Calculus of Mobile Processes, I and II. Inf. Comput. 100 (1), 1–77.

[100] Moser, G., Schnabl, A., 2011. The Derivational Complexity Induced by the Dependency Pair Method. Logical Methods in Computer Science 7 (3).

[101] Moser, G., Schnabl, A., Waldmann, J., 2008. Complexity Analysis of Term Rewriting Based on Matrix and Context Dependent Interpretations. In: Hariharan, R., Mukund, M., Vinay, V. (Eds.), IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2008, December 9-11, 2008, Bangalore, India. Vol. 2 of LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 304–315.

[102] Nagaya, T., March 1999. Reduction Strategies for Term Rewriting Systems. Ph.D. thesis, School of Information Science, Japan Advanced Institute of Science and Technology.

[103] Nakamura, M., March 2002. Reduction Strategies for Term Rewriting Systems. Ph.D. thesis, School of Information Science, Japan Advanced Institute of Science and Technology.

[104] Nakamura, M., Ogata, K., 2000. The evaluation strategy for head normal form with and without on-demand flags 36, 212–228.

[105] Nakamura, M., Ogata, K., Futatsugi, K., 2010. Reducibility of operation symbols in term rewriting systems and its application to behavioral specifications. J. Symb. Comput. 45 (5), 551–573.

[106] Neurauter, F., Zankl, H., Middeldorp, A., 2010. Revisiting Matrix Interpretations for Polynomial Derivational Complexity of Term Rewriting. In: Fermüller, C. G., Voronkov, A. (Eds.), Logic for Programming, Artificial Intelligence, and Reasoning - 17th International Conference, LPAR-17, Yogyakarta, Indonesia, October 10-15, 2010. Proceedings. Vol. 6397 of Lecture Notes in Computer Science. Springer, pp. 550–564.

[107] Nishida, N., Kuroda, T., Yanagisawa, M., Gmeiner, K., 2015. CO3: a COnverter for proving COnfluence of COnditional TRSs. In: 4th International Workshop on Confluence, IWC 2015, August 2, 2015, Berlin, Germany. p. 42.

[108] Nishida, N., Mizutani, T., Sakai, M., 2007. Transformation for Refining Unraveled Conditional Term Rewriting Systems. Electron. Notes Theor. Comput. Sci. 174 (10), 75–95.

[109] Nishida, N., Sakai, M., 2009. Completion after Program Inversion of Injective Functions. Electron. Notes Theor. Comput. Sci. 237, 39–56.

[110] Nishida, N., Sakai, M., Sakabe, T., 2005. Partial Inversion of Constructor Term Rewriting Systems. In: Giesl, J. (Ed.), Term Rewriting and Applications, 16th International Conference, RTA 2005, Nara, Japan, April 19-21, 2005, Proceedings. Vol. 3467 of Lecture Notes in Computer Science. Springer, pp. 264–278.

[111] Nishida, N., Sakai, M., Sakabe, T., 2011. Soundness of Unravelings for Deterministic Conditional Term Rewriting Systems via Ultra-Properties Related to Linearity. In: Schmidt-Schauß, M. (Ed.), Proceedings of the 22nd International Conference on Rewriting Techniques and Applications, RTA 2011, May 30 - June 1, 2011, Novi Sad, Serbia. Vol. 10 of LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 267–282.

[112] Nishida, N., Sakai, M., Sakabe, T., 2012. Soundness of Unravelings for Conditional Term Rewriting Systems via Ultra-Properties Related to Linearity. Log. Methods Comput. Sci. 8 (3).

[113] Nishida, N., Yanagisawa, M., Gmeiner, K., 2014. On Proving Confluence of Conditional Term Rewriting Systems via the Computationally Equivalent Transformation. In: 3rd International Workshop on Confluence, IWC 2014, July 13, 2014, Vienna, Austria. p. 42.

[114] Ogata, K., Futatsugi, K., 2000. Operational Semantics of Rewriting with the On-demand Evaluation Strategy. In: Bryant, B. R., Carroll, J. H., Damiani, E., Haddad, H., Oppenheim, D. (Eds.), Applied Computing 2000, Proceedings of the 2000 ACM Symposium on Applied Computing, Villa Olmo, Via Cantoni 1, 22100 Como, Italy, March 19-21, 2000. Volume 2. ACM, pp. 756–764.

[115] Ohlebusch, E., 1999. Transforming Conditional Rewrite Systems with Extra Variables into Unconditional Systems. In: Ganzinger, H., McAllester, D. A., Voronkov, A. (Eds.), Logic Programming and Automated Reasoning, 6th International Conference, LPAR'99, Tbilisi, Georgia, September 6-10, 1999, Proceedings. Vol. 1705 of Lecture Notes in Computer Science. Springer, pp. 111–130.

[116] Ohlebusch, E., 2002. Advanced topics in term rewriting. Springer.

[117] Ölveczky, P. C., Meseguer, J., 2002. Specification of real-time and hybrid systems in rewriting logic. Theor. Comput. Sci. 285 (2), 359–405.

[118] Ölveczky, P. C., Meseguer, J., 2004. Real-Time Maude 2.1 117, 285–314.

[119] Péchoux, R., 2013. Synthesis of sup-interpretations: A survey. Theor. Comput. Sci. 467, 30–52.

[120] Raffelsieper, M., Zantema, H., 2009. A Transformational Approach to Prove Outermost Termination Automatically. Electron. Notes Theor. Comput. Sci. 237, 3–21.

[121] Rubio, R., Martí-Oliet, N., Pita, I., Verdejo, A., 2018. Parameterized Strategies Specification in Maude. In: Fiadeiro, J. L., Tutu, I. (Eds.), Recent Trends in Algebraic Development Techniques - 24th IFIP WG 1.3 International Workshop, WADT 2018, Egham, UK, July 2-5, 2018, Revised Selected Papers. Vol. 11563 of Lecture Notes in Computer Science. Springer, pp. 27–44.

[122] Schernhammer, F., Gramlich, B., 2008. Termination of Lazy Rewriting Revisited. Electron. Notes Theor. Comput. Sci. 204, 35–51.

[123] Schernhammer, F., Gramlich, B., 2010. Characterizing and proving operational termination of deterministic conditional term rewriting systems. J. Log. Algebraic Methods Program. 79 (7), 659–688.

[124] Serbanuta, T., Rosu, G., 2006. Computationally Equivalent Elimination of Conditions. In: Pfenning, F. (Ed.), Term Rewriting and Applications, 17th International Conference, RTA 2006, Seattle, WA, USA, August 12-14, 2006, Proceedings. Vol. 4098 of Lecture Notes in Computer Science. Springer, pp. 19–34.

[125] Sijtsma, B. A., 1989. On the Productivity of Recursive List Definitions. ACM Trans. Program. Lang. Syst. 11 (4), 633–649.

[126] Slagle, J. R., 1974. Automated Theorem-Proving for Theories with Simplifiers Commutativity, and Associativity. J. ACM 21 (4), 622–642.

[127] Thati, P., Sen, K., Martí-Oliet, N., 2002. An Executable Specification of Asynchronous Pi-Calculus Semantics and May Testing in Maude 2.0. Electron. Notes Theor. Comput. Sci. 71, 261–281.

[128] Weiermann, A., 1995. Termination Proofs for Term Rewriting Systems by Lexicographic Path Orderings Imply Multiply Recursive Derivation Lengths. Theor. Comput. Sci. 139 (1&2), 355–362.

[129] Yamada, A., Sternagel, C., Thiemann, R., Kusakari, K., 2016. AC Dependency Pairs Revisited. In: Talbot, J., Regnier, L. (Eds.), 25th EACSL Annual Conference on Computer Science Logic, CSL 2016, August 29 - September 1, 2016, Marseille, France. Vol. 62 of LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 8:1–8:16.

[130] Zantema, H., 1995. Termination of Term Rewriting by Semantic Labelling. Fundam. Inform. 24 (1/2), 89–105.

[131] Zantema, H., Raffelsieper, M., 2009. Stream Productivity by Outermost Termination. In: Fernández, M. (Ed.), Proceedings Ninth International Workshop on Reduction Strategies in Rewriting and Programming, WRS 2009, Brasilia, Brazil, 28th June 2009. Vol. 15 of EPTCS. pp. 83–95.

[132] Zantema, H., Raffelsieper, M., 2010. Proving Productivity in Infinite Data Structures. In: Lynch, C. (Ed.), Proceedings of the 21st International Conference on Rewriting Techniques and Applications, RTA 2010, July 11-13, 2010, Edinburgh, Scottland, UK. Vol. 6 of LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 401–416.

Appendix A. Proof of $\mu^{\mathcal{U}}$-termination of $\mathcal{U}(\mathcal{R})$ in Example 7

Although AProVE and MU-TERM can be used for proving termination of *CSR*, we failed to obtain an automatic proof of $\mu^{\mathcal{U}}$-termination of $\mathcal{U}(\mathcal{R})$ in Example 7 with $\mu^{\mathcal{U}}$ in Example 11 by using the available versions of the tools. For this reason, in this appendix we develop a semi-automatic proof based on some recent developments.

First, according to [2], the $\mu^{\mathcal{U}}$-termination of $\mathcal{U}(\mathcal{R})$ is equivalent to the absence of infinite *chains* of *context-sensitive dependency pairs* (CSDPs). For $(\mathcal{U}(\mathcal{R}), \mu^{\mathcal{U}})$, we have the following CSDPs:

$$\begin{array}{rclll} \mathsf{G}(x) & \to & U^\sharp(\mathsf{f}(x), x) & \text{(A.1)} & \qquad U^\sharp(x,x) \to \mathsf{G}(\mathsf{a}) \quad \text{(A.3)} \\ \mathsf{G}(x) & \to & \mathsf{F}(x) & \text{(A.2)} & \qquad U^\sharp(x,x) \to \mathsf{A} \quad \text{(A.4)} \end{array}$$

which are obtained by just collecting in $\mathsf{DP}(\mathcal{R},\mu)$ (the set of CSDPs for $\mathcal{R}$), a rule $\ell^\sharp \to s^\sharp$ for each $\ell \to r \in \mathcal{U}(\mathcal{R})$, where s is an active subterm of r with $root(s) \in \mathcal{D}$ and for all terms $t = f(t_1, \ldots, t_k)$, $t^\sharp$ denotes the *marking* of t as $f^\sharp(t_1, \ldots, t_k)$ (i.e., only the root symbol f is marked in t). Note that the marked versions $f^\sharp$ of symbols $f \in \mathcal{D}$ (often just capitalized: F instead of $f^\sharp$) are assumed to be *different* from any other symbol in $\mathcal{F}$ (or previously introduced by marking).

Now, a *chain* of dependency pairs is a (finite or infinite sequence) $(u_i \to v_i)_{i\geq 1}$ where $u_i \to v_i$ are renamed versions of CSDPs $\mathsf{DP}(\mathcal{R},\mu)$ so that, for all i, j with $i \neq j$, $\mathcal{V}ar(u_i) \cap \mathcal{V}ar(u_j) = \varnothing$. Furthermore, there is a substitution σ such that, for all $i \geq 1$, $\sigma(v_i) \hookrightarrow^*_{\mathcal{R}} \sigma(u_{i+1})$.[22]

A proof of μ-termination using CSDPs typically starts with the construction of the *context-sensitive dependency graph* which is a graph $\mathsf{DG}(\mathcal{R},\mu)$ whose nodes are the elements of $\mathsf{DP}(\mathcal{R},\mu)$.

[22] These are simplified definitions of CSDPs and chains of CSDPs which nevertheless suffice to deal with our simple example. Further details can be found in [2].

Example 33. *The set of nodes of* $\mathsf{DG}(\mathcal{U}(\mathcal{R}), \mu^{\mathcal{U}})$ *is* $\{(A.1), (A.2), (A.3), (A.4)\}$.

There is an arc from $u \to v \in \mathsf{DP}(\mathcal{R}, \mu)$ to (a renamed version) $u' \to v'$ of a pair in $\mathsf{DP}(\mathcal{R}, \mu)$ iff (i) $\mathcal{V}ar(u) \cap \mathcal{V}ar(u') = \varnothing$ and (ii) $\sigma(v) \hookrightarrow^* \sigma(u')$ for some substitution σ.

Example 34. *There is an arc from (A.3) to (A.1) (and (A.2)) because* $v_{(A.3)} = \mathsf{G}(\mathsf{a})$ *is an instance of* $\mathsf{G}(x) = u_{(A.1)} = u_{(A.2)}$ *with* $\sigma(x) = \mathsf{a}$. *Thus, we have* $\sigma(v_{(A.3)}) = \mathsf{G}(\mathsf{a})) = \sigma(u_{(A.1)}) = \sigma(u_{(A.2)})$.

However, there is no arc from (A.2) to any other node in the graph because no lhs u *in a CSDP* $u \to v$ *is rooted with* F *and the symbol* F *in the right-hand side of* (A.2) *cannot be changed by rewritings with* $\mathcal{R}$ *as it is not in the signature* $\mathcal{F}$ *of* $\mathcal{R}$ *(marked symbols* $f^\sharp$ *are different from any other symbol in* $\mathcal{F}$*). Similarly, there is no arc from (A.4) to any other node in* $\mathsf{DG}(\mathcal{U}(\mathcal{R}), \mu^{\mathcal{U}})$.

The graph is intended to represent chains of CSDPs as *paths* in the graph. An important fact is that the absence of *cycles* in $\mathsf{DG}(\mathcal{R}, \mu)$ implies the μ-termination of $\mathcal{R}$. Thus, what we do in the following is just showing that there is no cycle in $\mathsf{DG}(\mathcal{U}(\mathcal{R}), \mu^{\mathcal{U}})$. In Example 34 we have enumerated all arcs outcoming from (A.2), (A.3), and (A.4). Regarding (A.1), we need to consider the following *feasibility goal* [59]:

$$U^\sharp(f(x), x) \hookrightarrow^* U^\sharp(y, y) \tag{A.5}$$

whose *infeasibility* can be automatically proved by using the tool $\mathsf{infChecker}$. If we obtain a model $\mathcal{A}$ of the theory $\overline{\mathcal{U}(\mathcal{R})^{\mu^{\mathcal{U}}}}$ associated to $(\mathcal{U}(\mathcal{R}), \mu^{\mathcal{U}})$, see Section 3, together with $\neg(\exists x, y)\ U^\sharp(f(x), x) \hookrightarrow^* U^\sharp(y, y)$, i.e., if

$$\mathcal{A} \models \overline{\mathcal{U}(\mathcal{R})^{\mu^{\mathcal{U}}}} \cup \{\neg(\exists x, y)\ U^\sharp(f(x), x) \hookrightarrow^* U^\sharp(y, y)\}$$

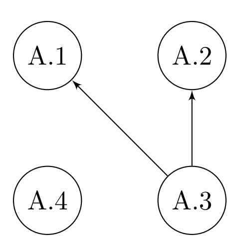
holds for some structure $\mathcal{A}$, then (A.5) is infeasible [59]. The following model is obtained by $\mathsf{infChecker}$: the domain is $\mathcal{A} = \mathbb{Z}$; for function and predicate symbols we have:

$$\mathsf{a}^{\mathcal{A}} = -1 \qquad \mathsf{b}^{\mathcal{A}} = 0 \qquad \mathsf{f}^{\mathcal{A}}(x) = x + 1$$

$$\mathsf{g}^{\mathcal{A}}(x) = 0 \qquad u^{\mathcal{A}}(x, y) = 0 \qquad U^{\mathcal{A}}(x, y) = x - y$$

$$x(\to_{\mathcal{R}})^{\mathcal{A}} y \Leftrightarrow y \geq x \wedge x + 1 \geq y \qquad x(\to_{\mathcal{R}}^*)^{\mathcal{A}} y \Leftrightarrow y \geq x$$

This witnesses that there is no arc from (A.1) to any other node. Overall, $\mathsf{DG}(\mathcal{U}(\mathcal{R}), \mu^{\mathcal{U}})$ is as follows:



Since there is no cycle in $\mathsf{DG}(\mathcal{U}(\mathcal{R}), \mu^{\mathcal{U}})$, we conclude $\mu^{\mathcal{U}}$-termination of $\mathcal{U}(\mathcal{R})$.