Reversible Computation in Term Rewriting[☆]

Naoki Nishida[a], Adrián Palacios[b], Germán Vidal[b,*]

[a] *Graduate School of Informatics, Nagoya University*
Furo-cho, Chikusa-ku, 4648603 Nagoya, Japan
[b] *MiST, DSIC, Universitat Politècnica de València*
Camino de Vera, s/n, 46022 Valencia, Spain

Abstract

Essentially, in a reversible programming language, for each forward computation from state S to state S', there exists a constructive method to go backwards from state S' to state S. Besides its theoretical interest, reversible computation is a fundamental concept which is relevant in many different areas like cellular automata, bidirectional program transformation, or quantum computing, to name a few.

In this work, we focus on term rewriting, a computation model that underlies most rule-based programming languages. In general, term rewriting is not reversible, even for injective functions; namely, given a rewrite step $t_1 \to t_2$, we do not always have a decidable method to get t_1 from t_2. Here, we introduce a conservative extension of term rewriting that becomes reversible. Furthermore, we also define two transformations, injectivization and inversion, to make a rewrite system reversible using standard term rewriting. We illustrate the usefulness of our transformations in the context of bidirectional program transformation.

[☆]This work has been partially supported by the EU (FEDER) and the Spanish *Ministerio de Economía y Competitividad* (MINECO) under grants TIN2013-44742-C4-1-R and TIN2016-76843-C4-1-R, by the *Generalitat Valenciana* under grant PROMETEO-II/2015/013 (SmartLogic), and by the COST Action IC1405 on Reversible Computation - extending horizons of computing. Adrián Palacios was partially supported by the EU (FEDER) and the Spanish *Ayudas para contratos predoctorales para la formación de doctores* and *Ayudas a la movilidad predoctoral para la realización de estancias breves en centros de I+D*, MINECO (SEIDI), under FPI grants BES-2014-069749 and EEBB-I-16-11469. Part of this research was done while the second and third authors were visiting Nagoya University; they gratefully acknowledge their hospitality.

[*]Corresponding author.
Email addresses: `nishida@i.nagoya-u.ac.jp` (Naoki Nishida), `apalacios@dsic.upv.es` (Adrián Palacios), `gvidal@dsic.upv.es` (Germán Vidal)

Published in the *Journal of Logical and Algebraic Methods in Programming*, vol. 94, pages 128-149, 2018.
DOI `https://doi.org/10.1016/j.jlamp.2017.10.003`.

Keywords: term rewriting, reversible computation, program transformation

1. Introduction

The notion of reversible computation can be traced back to Landauer's pioneering work [22]. Although Landauer was mainly concerned with the energy consumption of erasing data in irreversible computing, he also claimed that every computer can be made reversible by saving the *history* of the computation. However, as Landauer himself pointed out, this would only postpone the problem of erasing the tape of a reversible Turing machine before it could be reused. Bennett [6] improved the original proposal so that the computation now ends with a tape that only contains the output of a computation and the initial source, thus deleting all remaining "garbage" data, though it performs twice the usual computation steps. More recently, Bennett's result is extended in [9] to nondeterministic Turing machines, where it is also proved that transforming an irreversible Turing machine into a reversible one can be done with a quadratic loss of space. We refer the interested reader to, e.g., [7, 14, 40] for a high level account of the principles of reversible computation.

In the last decades, reversible computing and *reversibilization* (transforming an irreversible computation device into a reversible one) have been the subject of intense research, giving rise to successful applications in many different fields, e.g., cellular automata [28], where reversibility is an essential property, bidirectional program transformation [24], where reversibility helps to automate the generation of inverse functions (see Section 6), reversible debugging [17], where one can go both forward and backward when seeking the cause of an error, parallel discrete event simulation [34], where reversible computation is used to undo the effects of speculative computations made on a wrong assumption, quantum computing [39], where all computations should be reversible, and so forth. The interested reader can find detailed surveys in the *state of the art* reports of the different working groups of COST Action IC1405 on Reversible Computation [20].

In this work, we introduce reversibility in the context of *term rewriting* [4, 36], a computation model that underlies most rule-based programming languages. In contrast to other, more *ad-hoc* approaches, we consider that

term rewriting is an excellent framework to rigorously define reversible computation in a functional context and formally prove its main properties. We expect our work to be useful in different (sequential) contexts, like reversible debugging, parallel discrete event simulation or bidirectional program transformation, to name a few. In particular, Section 6 presents a first approach to formalize bidirectional program transformation in our setting.

To be more precise, we present a general and intuitive notion of *reversible* term rewriting by defining a Landauer embedding. Given a rewrite system $\mathcal{R}$ and its associated (standard) rewrite relation $\to_{\mathcal{R}}$, we define a reversible extension of rewriting with two components: a forward relation $\rightharpoonup_{\mathcal{R}}$ and a backward relation $\leftharpoondown_{\mathcal{R}}$, such that $\rightharpoonup_{\mathcal{R}}$ is a conservative extension of $\to_{\mathcal{R}}$ and, moreover, $(\rightharpoonup_{\mathcal{R}})^{-1} = \leftharpoondown_{\mathcal{R}}$. We note that the inverse rewrite relation, $(\to_{\mathcal{R}})^{-1}$, is not an appropriate basis for "reversible" rewriting since we aim at defining a technique to *undo* a particular reduction. In other words, given a rewriting reduction $s \to^*_{\mathcal{R}} t$, our reversible relation aims at computing the term s from t and $\mathcal{R}$ in a decidable and deterministic way, which is not possible using $(\to_{\mathcal{R}})^{-1}$ since it is generally non-deterministic, non-confluent, and non-terminating, even for systems defining injective functions (see Example 6). In contrast, our backward relation $\leftharpoondown_{\mathcal{R}}$ is deterministic (thus confluent) and terminating. Moreover, our relation proceeds backwards step by step, i.e., the number of reduction steps in $s \rightharpoonup^*_{\mathcal{R}} t$ and $t \leftharpoondown^*_{\mathcal{R}} s$ are the same.

In order to introduce a reversibilization transformation for rewrite systems, we use a *flattening* transformation so that the reduction at top positions of terms suffices to get a normal form in the transformed systems. For instance, given the following rewrite system:

$$\begin{array}{rcl} \mathsf{add}(\mathsf{0}, y) & \to & y, \\ \mathsf{add}(\mathsf{s}(x), y) & \to & \mathsf{s}(\mathsf{add}(x, y)) \end{array}$$

defining the addition on natural numbers built from constructors $\mathsf{0}$ and $\mathsf{s}(\ )$, we produce the following *flattened* (conditional) system:

$$\begin{array}{rcl} \mathcal{R} = \{ \quad \mathsf{add}(\mathsf{0}, y) & \to & y, \\ \mathsf{add}(\mathsf{s}(x), y) & \to & \mathsf{s}(z) \Leftarrow \mathsf{add}(x, y) \twoheadrightarrow z \quad \} \end{array}$$

(see Example 29 for more details). This allows us to provide an improved notion of reversible rewriting in which some information (namely, the positions where reduction takes place) is not required anymore. This opens the door to *compile* the reversible extension of rewriting into the system rules. Loosely speaking, given a system $\mathcal{R}$, we produce new systems $\mathcal{R}_f$ and $\mathcal{R}_b$ such that *standard* rewriting in $\mathcal{R}_f$, i.e., $\to_{\mathcal{R}_f}$, coincides with the forward reversible extension $\rightharpoonup_{\mathcal{R}}$ in the original system, and analogously $\to_{\mathcal{R}_b}$ is equivalent to

$\leftharpoondown_{\mathcal{R}}$. E.g., for the system $\mathcal{R}$ above, we would produce

$$\begin{aligned}\mathcal{R}_f = \{ \quad & \mathsf{add}^{\mathtt{i}}(\mathsf{0}, y) \to \langle y, \beta_1 \rangle, \\ & \mathsf{add}^{\mathtt{i}}(\mathsf{s}(x), y) \to \langle \mathsf{s}(z), \beta_2(w) \rangle \Leftarrow \mathsf{add}^{\mathtt{i}}(x, y) \twoheadrightarrow \langle z, w \rangle \ \}\end{aligned}$$

$$\begin{aligned}\mathcal{R}_b = \{ \quad & \mathsf{add}^{-1}(y, \beta_1) \to \langle \mathsf{0}, y \rangle, \\ & \mathsf{add}^{-1}(\mathsf{s}(z), \beta_2(w)) \to \langle \mathsf{s}(x), y \rangle \Leftarrow \mathsf{add}^{-1}(z, w) \to \langle x, y \rangle \quad \ \}\end{aligned}$$

where $\mathsf{add}^{\mathtt{i}}$ is an injective version of function add, add^{-1} is the inverse of $\mathsf{add}^{\mathtt{i}}$, and β_1, β_2 are fresh symbols introduced to label the rules of $\mathcal{R}$.

In this work, we will mostly consider *conditional* rewrite systems, not only to have a more general notion of reversible rewriting, but also to define a reversibilization technique for unconditional rewrite systems, since the application of *flattening* (cf. Section 4) may introduce conditions in a system that is originally unconditional, as illustrated above.

This paper is an extended version of [31]. In contrast to [31], our current paper includes the proofs of technical results, the reversible extension of term rewriting is introduced first in the unconditional case (which is simpler and more intuitive), and presents an improved injectivization transformation when the system includes injective functions. Furthermore, a prototype implementation of the reversibilization technique is publicly available from `http://kaz.dsic.upv.es/rev-rewriting.html`.

The paper is organized as follows. After introducing some preliminaries in Section 2, we present our approach to reversible term rewriting in Section 3. Section 4 introduces the class of *pure constructor* systems where all reductions take place at topmost positions, so that storing this information in reversible rewrite steps becomes unnecessary. Then, Section 5 presents injectivization and inversion transformations in order to make a rewrite system reversible with standard rewriting. Here, we also present an improvement of the transformation for injective functions. The usefulness of these transformations is illustrated in Section 6. Finally, Section 7 discusses some related work and Section 8 concludes and points out some ideas for future research.

2. Preliminaries

We assume familiarity with basic concepts of term rewriting. We refer the reader to, e.g., [4] and [36] for further details.

2.1. Terms and Substitutions

A *signature* $\mathcal{F}$ is a set of ranked function symbols. Given a set of variables $\mathcal{V}$ with $\mathcal{F} \cap \mathcal{V} = \varnothing$, we denote the domain of *terms* by $\mathcal{T}(\mathcal{F}, \mathcal{V})$. We use $\mathsf{f}, \mathsf{g}, \ldots$ to denote functions and $x, y, \ldots$ to denote variables. Positions are used to

address the nodes of a term viewed as a tree. A *position* p in a term t, in symbols $p \in \mathcal{P}\mathsf{os}(t)$, is represented by a finite sequence of natural numbers, where ϵ denotes the root position. We let $t|_p$ denote the *subterm* of t at position p and $t[s]_p$ the result of *replacing the subterm* $t|_p$ by the term s. $\mathcal{V}ar(t)$ denotes the set of variables appearing in t. We also let $\mathcal{V}ar(t_1, \ldots, t_n)$ denote $\mathcal{V}ar(t_1) \cup \cdots \cup \mathcal{V}ar(t_n)$. A term t is *ground* if $\mathcal{V}ar(t) = \varnothing$.

A *substitution* $\sigma : \mathcal{V} \mapsto \mathcal{T}(\mathcal{F}, \mathcal{V})$ is a mapping from variables to terms such that $\mathcal{D}\mathsf{om}(\sigma) = \{x \in \mathcal{V} \mid x \neq \sigma(x)\}$ is its domain. A substitution σ is *ground* if $x\sigma$ is ground for all $x \in \mathcal{D}\mathsf{om}(\sigma)$. Substitutions are extended to morphisms from $\mathcal{T}(\mathcal{F}, \mathcal{V})$ to $\mathcal{T}(\mathcal{F}, \mathcal{V})$ in the natural way. We denote the application of a substitution σ to a term t by $t\sigma$ rather than $\sigma(t)$. The identity substitution is denoted by *id*. We let "$\circ$" denote the composition of substitutions, i.e., $\sigma \circ \theta(x) = (x\theta)\sigma = x\theta\sigma$. The *restriction* $\theta\restriction_V$ of a substitution θ to a set of variables V is defined as follows: $x\theta\restriction_V = x\theta$ if $x \in V$ and $x\theta\restriction_V = x$ otherwise.

2.2. Term Rewriting Systems

A set of rewrite rules $l \to r$ such that l is a nonvariable term and r is a term whose variables appear in l is called a *term rewriting system* (TRS for short); terms l and r are called the left-hand side and the right-hand side of the rule, respectively. We restrict ourselves to finite signatures and TRSs. Given a TRS $\mathcal{R}$ over a signature $\mathcal{F}$, the *defined* symbols $\mathcal{D}_\mathcal{R}$ are the root symbols of the left-hand sides of the rules and the *constructors* are $\mathcal{C}_\mathcal{R} = \mathcal{F} \setminus \mathcal{D}_\mathcal{R}$. *Constructor terms* of $\mathcal{R}$ are terms over $\mathcal{C}_\mathcal{R}$ and $\mathcal{V}$, denoted by $\mathcal{T}(\mathcal{C}_\mathcal{R}, \mathcal{V})$. We sometimes omit $\mathcal{R}$ from $\mathcal{D}_\mathcal{R}$ and $\mathcal{C}_\mathcal{R}$ if it is clear from the context. A substitution σ is a *constructor substitution* (of $\mathcal{R}$) if $x\sigma \in \mathcal{T}(\mathcal{C}_\mathcal{R}, \mathcal{V})$ for all variables x.

For a TRS $\mathcal{R}$, we define the associated rewrite relation $\to_\mathcal{R}$ as the smallest binary relation on terms satisfying the following: given terms $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, we have $s \to_\mathcal{R} t$ iff there exist a position p in s, a rewrite rule $l \to r \in \mathcal{R}$, and a substitution σ such that $s|_p = l\sigma$ and $t = s[r\sigma]_p$; the rewrite step is sometimes denoted by $s \to_{p,l\to r} t$ to make explicit the position and rule used in this step. The instantiated left-hand side $l\sigma$ is called a *redex*. A term s is called *irreducible* or in *normal form* with respect to a TRS $\mathcal{R}$ if there is no term t with $s \to_\mathcal{R} t$. A substitution is called *normalized* with respect to $\mathcal{R}$ if every variable in the domain is replaced by a normal form with respect to $\mathcal{R}$. We sometimes omit "with respect to $\mathcal{R}$" if it is clear from the context. A *derivation* is a (possibly empty) sequence of rewrite steps. Given a binary relation $\to$, we denote by $\to^*$ its reflexive and transitive closure, i.e., $s \to^*_\mathcal{R} t$ means that s can be reduced to t in $\mathcal{R}$ in zero or more steps; we also use $s \to^n_\mathcal{R} t$ to denote that s can be reduced to t in exactly n steps.

We further assume that rewrite rules are labelled, i.e., given a TRS $\mathcal{R}$, we denote by $\beta : l \to r$ a rewrite rule with label β. Labels are unique in a TRS. Also, to relate label β to fixed variables, we consider that the variables of the rewrite rules are not renamed[1] and that the reduced terms are always ground. Equivalently, one could require terms to be variable disjoint with the variables of the rewrite system, but we require groundness for simplicity. We often write $s \to_{p,\beta} t$ instead of $s \to_{p,l\to r} t$ if rule $l \to r$ is labeled with β.

2.3. Conditional Term Rewrite Systems

In this paper, we also consider *conditional* term rewrite systems (CTRSs); namely oriented 3-CTRSs, i.e., CTRSs where extra variables are allowed as long as $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l) \cup \mathcal{V}ar(C)$ for any rule $l \to r \Leftarrow C$ [26]. In *oriented* CTRSs, a conditional rule $l \to r \Leftarrow C$ has the form $l \to r \Leftarrow s_1 \twoheadrightarrow t_1, \ldots, s_n \twoheadrightarrow t_n$, where each oriented equation $s_i \twoheadrightarrow t_i$ is interpreted as reachability ($\to_{\mathcal{R}}^*$). In the following, we denote by $\overline{o_n}$ a sequence of elements $o_1, \ldots, o_n$ for some n. We also write $\overline{o_{i,j}}$ for the sequence $o_i, \ldots, o_j$ when $i \leq j$ (and the empty sequence otherwise). We write $\overline{o}$ when the number of elements is not relevant. In addition, we denote a condition $o_1 \twoheadrightarrow o'_1, \ldots, o_n \twoheadrightarrow o'_n$ by $\overline{o_n \twoheadrightarrow o'_n}$.

As in the unconditional case, we consider that rules are labelled and that labels are unique in a CTRS. And, again, to relate label β to fixed variables, we consider that the variables of the conditional rewrite rules are not renamed and that the reduced terms are always ground.

For a CTRS $\mathcal{R}$, the associated rewrite relation $\to_{\mathcal{R}}$ is defined as the smallest binary relation satisfying the following: given ground terms $s, t \in \mathcal{T}(\mathcal{F})$, we have $s \to_{\mathcal{R}} t$ iff there exist a position p in s, a rewrite rule $l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n} \in \mathcal{R}$, and a ground substitution σ such that $s|_p = l\sigma$, $s_i\sigma \to_{\mathcal{R}}^* t_i\sigma$ for all $i = 1, \ldots, n$, and $t = s[r\sigma]_p$.

In order to simplify the presentation, we only consider *deterministic* CTRSs (DCTRSs), i.e., oriented 3-CTRSs where, for each rule $l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n}$, we have $\mathcal{V}\mathsf{ar}(s_i) \subseteq \mathcal{V}\mathsf{ar}(l, \overline{t_{i-1}})$ for all $i = 1, \ldots, n$ (see Section 3.2 for a justification of this requirement and how it could be relaxed to arbitrary 3-CTRSs). Intuitively speaking, the use of DCTRs allows us to compute the bindings for the variables in the condition of a rule in a deterministic way. E.g., given a ground term s and a rule $\beta : l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n}$ with $s|_p = l\theta$, we have that $s_1\theta$ is ground. Therefore, one can reduce $s_1\theta$ to some term s'_1 such that s'_1 is an instance of $t_1\theta$ with some ground substitution θ_1. Now, we have

[1] This will become useful in the next section where the reversible extension of rewriting keeps a "history" of a computation in the form of a list of terms $\beta(p, \sigma)$, and we want the domain of σ to be a subset of the left-hand side of the rule labelled with β.

that $s_2\theta\theta_1$ is ground and we can reduce $s_2\theta\theta_1$ to some term s_2' such that s_2' is an instance of $t_2\theta\theta_1$ with some ground substitution θ_2, and so forth. If all equations in the condition hold using $\theta_1, \ldots, \theta_n$, we have that $s \to_{p,\beta} s[r\sigma]_p$ with $\sigma = \theta\theta_1 \ldots \theta_n$.

Example 1. Consider the following DCTRS $\mathcal{R}$ that defines the function double that doubles the value of its argument when it is an even natural number:

$$\begin{array}{ll} \beta_1 : \quad \mathsf{add}(0, y) \to y & \beta_4 : \quad \mathsf{even}(0) \to \mathsf{true} \\ \beta_2 : \mathsf{add}(\mathsf{s}(x), y) \to \mathsf{s}(\mathsf{add}(x, y)) & \beta_5 : \mathsf{even}(\mathsf{s}(\mathsf{s}(x))) \to \mathsf{even}(x) \\ \beta_3 : \quad \mathsf{double}(x) \to \mathsf{add}(x, x) \Leftarrow \mathsf{even}(x) \twoheadrightarrow \mathsf{true} & \end{array}$$

Given the term $\mathsf{double}(\mathsf{s}(\mathsf{s}(0)))$ we have, for instance, the following derivation:

$$\begin{array}{lll} \mathsf{double}(\mathsf{s}(\mathsf{s}(0))) \to_{\epsilon,\beta_3} & \mathsf{add}(\mathsf{s}(\mathsf{s}(0)), \mathsf{s}(\mathsf{s}(0))) & \text{since } \mathsf{even}(\mathsf{s}(\mathsf{s}(0))) \to_{\mathcal{R}}^* \mathsf{true} \\ & & \text{with } \sigma = \{x \mapsto \mathsf{s}(\mathsf{s}(0))\} \\ \to_{\epsilon,\beta_2} & \mathsf{s}(\mathsf{add}(\mathsf{s}(0), \mathsf{s}(\mathsf{s}(0)))) & \text{with } \sigma = \{x \mapsto \mathsf{s}(0),\ y \mapsto \mathsf{s}(\mathsf{s}(0))\} \\ \to_{1,\beta_2} & \mathsf{s}(\mathsf{s}(\mathsf{add}(0, \mathsf{s}(\mathsf{s}(0))))) & \text{with } \sigma = \{x \mapsto 0, y \mapsto \mathsf{s}(\mathsf{s}(0))\} \\ \to_{1.1,\beta_1} & \mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(0)))) & \text{with } \sigma = \{y \mapsto \mathsf{s}(\mathsf{s}(0))\} \end{array}$$

3. Reversible Term Rewriting

In this section, we present a conservative extension of the rewrite relation which becomes reversible. In the following, we use $\rightharpoonup_{\mathcal{R}}$ to denote our *reversible* (forward) term rewrite relation, and $\leftharpoondown_{\mathcal{R}}$ to denote its application in the reverse (backward) direction. Note that, in principle, we do not require $\leftharpoondown_{\mathcal{R}} = \rightharpoonup_{\mathcal{R}}^{-1}$, i.e., we provide independent (constructive) definitions for each relation. Nonetheless, we will prove that $\leftharpoondown_{\mathcal{R}} = \rightharpoonup_{\mathcal{R}}^{-1}$ indeed holds (cf. Theorems 9 and 20). In some approaches to reversible computing, both forward and backward relations should be deterministic. Here, we will only require deterministic *backward* steps, while forward steps might be non-deterministic, as it is often the case in term rewriting.

3.1. Unconditional Term Rewrite Systems

We start with unconditional TRSs since it is conceptually simpler and thus will help the reader to better understand the key ingredients of our approach. In the next section, we will consider the more general case of DCTRSs.

Given a TRS $\mathcal{R}$, reversible rewriting is defined on pairs $\langle t, \pi \rangle$, where t is a ground term and π is a trace (the "history" of the computation so far). Here, a *trace* in $\mathcal{R}$ is a list of *trace terms* of the form $\beta(p, \sigma)$ such that β is

a label for some rule $l \to r \in \mathcal{R}$, p is a position, and σ is a substitution with $\mathcal{D}\mathsf{om}(\sigma) = \mathcal{V}\mathsf{ar}(l) \backslash \mathcal{V}\mathsf{ar}(r)$ which will record the bindings of erased variables when $\mathcal{V}\mathsf{ar}(l) \backslash \mathcal{V}\mathsf{ar}(r) \neq \varnothing$ (and $\sigma = id$ if $\mathcal{V}\mathsf{ar}(l) \backslash \mathcal{V}\mathsf{ar}(r) = \varnothing$).[2] Our trace terms have some similarities with *proof terms* [36]. However, proof terms do not store the bindings of erased variables (and, to the best of our knowledge, they are only defined for unconditional TRSs, while we use trace terms both for unconditional and conditional TRSs).

Our reversible term rewriting relation is only defined on *safe* pairs:

Definition 2. Let $\mathcal{R}$ be a TRS. The pair $\langle s, \pi \rangle$ is *safe* in $\mathcal{R}$ iff, for all $\beta(p, \sigma)$ in π, σ is a ground substitution with $\mathcal{D}\mathsf{om}(\sigma) = \mathcal{V}\mathsf{ar}(l) \backslash \mathcal{V}\mathsf{ar}(r)$ and $\beta : l \to r \in \mathcal{R}$.

In the following, we often omit $\mathcal{R}$ when referring to traces and safe pairs if the underlying TRS is clear from the context.

Safety is not necessary when applying a forward reduction step, but will become essential for the backward relation $\leftharpoondown_{\mathcal{R}}$ to be correct. E.g., all traces that come from the forward reduction of some initial pair with an empty trace will be safe (see below). Reversible rewriting is then introduced as follows:

Definition 3. Let $\mathcal{R}$ be a TRS. A reversible rewrite relation $\rightharpoonup_{\mathcal{R}}$ is defined on safe pairs $\langle t, \pi \rangle$, where t is a ground term and π is a trace in $\mathcal{R}$. The reversible rewrite relation extends standard rewriting as follows:[3]

$$\langle s, \pi \rangle \rightharpoonup_{\mathcal{R}} \langle t, \beta(p, \sigma') : \pi \rangle$$

iff there exist a position $p \in \mathcal{P}\mathsf{os}(s)$, a rewrite rule $\beta : l \to r \in \mathcal{R}$, and a ground substitution σ such that $s|_p = l\sigma$, $t = s[r\sigma]_p$, and $\sigma' = \sigma|_{\mathcal{V}\mathsf{ar}(l) \backslash \mathcal{V}\mathsf{ar}(r)}$. The reverse relation, $\leftharpoondown_{\mathcal{R}}$, is then defined as follows:

$$\langle t, \beta(p, \sigma') : \pi \rangle \leftharpoondown_{\mathcal{R}} \langle s, \pi \rangle$$

iff $\langle t, \beta(p, \sigma') : \pi \rangle$ is a safe pair in $\mathcal{R}$ and there exist a ground substitution θ and a rule $\beta : l \to r \in \mathcal{R}$ such that $\mathcal{D}\mathsf{om}(\theta) = \mathcal{V}\mathsf{ar}(r)$, $t|_p = r\theta$ and $s = t[l\theta\sigma']_p$. Note that $\theta\sigma' = \sigma'\theta = \theta \cup \sigma'$, where $\cup$ is the union of substitutions, since $\mathcal{D}\mathsf{om}(\theta) = \mathcal{V}\mathsf{ar}(r)$, $\mathcal{D}\mathsf{om}(\sigma') = (\mathcal{V}\mathsf{ar}(l) \backslash \mathcal{V}\mathsf{ar}(r))$ and both substitutions are ground, so $\mathcal{D}\mathsf{om}(\theta) \cap \mathcal{D}\mathsf{om}(\sigma') = \varnothing$.

[2] Note that if a rule $l \to r$ is non-erasing, i.e., $\mathcal{V}\mathsf{ar}(l) = \mathcal{V}\mathsf{ar}(r)$, then $\sigma = id$.

[3] In the following, we consider the usual infix notation for lists where $[\,]$ is the empty list and $x : xs$ is a list with head x and tail xs.

We denote the union of both relations $\rightharpoonup_{\mathcal{R}} \cup \leftharpoondown_{\mathcal{R}}$ by $\rightleftharpoons_{\mathcal{R}}$.

Example 4. Let us consider the following TRS $\mathcal{R}$ defining the addition on natural numbers built from $\mathsf{0}$ and $\mathsf{s}(\ )$, and the function fst that returns its first argument:

$$\begin{array}{lrcl} \beta_1: & \mathsf{add}(\mathsf{0}, y) & \to & y \\ \beta_2: & \mathsf{add}(\mathsf{s}(x), y) & \to & \mathsf{s}(\mathsf{add}(x, y)) \end{array} \qquad \beta_3: \ \mathsf{fst}(x, y) \to x$$

Given the term $\mathsf{fst}(\mathsf{add}(\mathsf{s}(\mathsf{0}), \mathsf{0}), \mathsf{0})$, we have, for instance, the following reversible (forward) derivation:

$$\begin{array}{rcl} \langle \mathsf{fst}(\mathsf{add}(\mathsf{s}(\mathsf{0}), \mathsf{0}), \mathsf{0}), [\,] \rangle & \rightharpoonup_{\mathcal{R}} & \langle \mathsf{fst}(\mathsf{s}(\mathsf{add}(\mathsf{0}, \mathsf{0})), \mathsf{0}), [\beta_2(1, id)] \rangle \\ & \rightharpoonup_{\mathcal{R}} & \langle \mathsf{s}(\mathsf{add}(\mathsf{0}, \mathsf{0})), [\beta_3(\epsilon, \{y \mapsto \mathsf{0}\}), \beta_2(1, id)] \rangle \\ & \rightharpoonup_{\mathcal{R}} & \langle \mathsf{s}(\mathsf{0}), [\beta_1(1, id), \beta_3(\epsilon, \{y \mapsto \mathsf{0}\}), \beta_2(1, id)] \rangle \end{array}$$

The reader can easily check that $\langle \mathsf{s}(\mathsf{0}), [\beta_1(1, id), \beta_3(\epsilon, \{y \mapsto \mathsf{0}\}), \beta_2(1, id)] \rangle$ is reducible to $\langle \mathsf{fst}(\mathsf{add}(\mathsf{s}(\mathsf{0}), \mathsf{0}), \mathsf{0}), [\,] \rangle$ using the backward relation $\leftharpoondown_{\mathcal{R}}$ by performing exactly the same steps but in the backward direction.

An easy but essential property of $\rightharpoonup_{\mathcal{R}}$ is that it is a conservative extension of standard rewriting in the following sense (we omit its proof since it is straightforward):

Theorem 5. *Let $\mathcal{R}$ be a TRS. Given terms s, t, if $s \to_{\mathcal{R}}^* t$, then for any trace π there exists a trace π' such that $\langle s, \pi \rangle \rightharpoonup_{\mathcal{R}}^* \langle t, \pi' \rangle$.*

Here, and in the following, we assume that $\leftarrow_{\mathcal{R}} = (\to_{\mathcal{R}})^{-1}$, i.e., $s \to_{\mathcal{R}}^{-1} t$ is denoted by $s \leftarrow_{\mathcal{R}} t$. Observe that the backward relation is not a conservative extension of $\leftarrow_{\mathcal{R}}$: in general, $t \leftarrow_{\mathcal{R}} s$ does not imply $\langle t, \pi' \rangle \leftharpoondown_{\mathcal{R}} \langle s, \pi \rangle$ for any arbitrary trace π'. This is actually the purpose of our notion of reversible rewriting: $\leftharpoondown_{\mathcal{R}}$ should not extend $\leftarrow_{\mathcal{R}}$ but is only aimed at performing *exactly the same steps* of the forward computation whose trace was stored, but in the reverse order. Nevertheless, one can still ensure that for all steps $t \leftarrow_{\mathcal{R}} s$, there exists some trace π' such that $\langle t, \pi' \rangle \leftharpoondown_{\mathcal{R}} \langle s, \pi \rangle$ (which is an easy consequence of the above result and Theorem 9 below).

Example 6. Consider again the following TRS $\mathcal{R} = \{\beta: \ \mathsf{snd}(x, y) \to y\}$. Given the reduction $\mathsf{snd}(1, 2) \to_{\mathcal{R}} 2$, there are infinitely many reductions for 2 using $\leftarrow_{\mathcal{R}}$, e.g., $2 \leftarrow_{\mathcal{R}} \mathsf{snd}(1, 2)$, $2 \leftarrow_{\mathcal{R}} \mathsf{snd}(2, 2)$, $2 \leftarrow_{\mathcal{R}} \mathsf{snd}(3, 2)$, etc. The relation is also non-terminating: $2 \leftarrow_{\mathcal{R}} \mathsf{snd}(1, 2) \leftarrow_{\mathcal{R}} \mathsf{snd}(1, \mathsf{snd}(1, 2)) \leftarrow_{\mathcal{R}} \cdots$. In contrast, given a pair $\langle 2, \pi \rangle$, we can only perform a single deterministic and finite reduction (as proved below). For instance, if $\pi = [\beta(\epsilon, \{x \mapsto \mathsf{1}\}), \beta(2, \{x \mapsto \mathsf{1}\})]$, then the only possible reduction is $\langle 2, \pi \rangle \leftharpoondown_{\mathcal{R}} \langle \mathsf{snd}(1, 2), [\beta(2, \{x \mapsto \mathsf{1}\})] \rangle \leftharpoondown_{\mathcal{R}} \langle \mathsf{snd}(1, \mathsf{snd}(1, 2)), [\,] \rangle \not\leftharpoondown_{\mathcal{R}}$.

Now, we state a lemma which shows that safe pairs are preserved through reversible term rewriting (both in the forward and backward directions):

Lemma 7. *Let $\mathcal{R}$ be a TRS. Let $\langle s, \pi\rangle$ be a safe pair. If $\langle s, \pi\rangle \rightleftharpoons_{\mathcal{R}}^{*} \langle t, \pi'\rangle$, then $\langle t, \pi'\rangle$ is also safe.*

Proof. We prove the claim by induction on the length k of the derivation. Since the base case $k = 0$ is trivial, consider the inductive case $k > 0$. Assume a derivation of the form $\langle s, \pi\rangle \rightleftharpoons_{\mathcal{R}}^{*} \langle s_0, \pi_0\rangle \rightleftharpoons_{\mathcal{R}} \langle t, \pi'\rangle$. By the induction hypothesis, we have that $\langle s_0, \pi_0\rangle$ is a safe pair. Now, we distinguish two cases depending on the last step. If we have $\langle s_0, \pi_0\rangle \rightharpoonup_{\mathcal{R}} \langle t, \pi'\rangle$, then there exist a position $p \in \mathcal{P}\mathsf{os}(s_0)$, a rewrite rule $\beta : l \to r \in \mathcal{R}$, and a ground substitution σ such that $s_0|_p = l\sigma$, $t = s_0[r\sigma]_p$, $\sigma' = \sigma\restriction_{\mathcal{V}\mathsf{ar}(l)\setminus\mathcal{V}\mathsf{ar}(r)}$, and $\pi' = \beta(p, \sigma') : \pi_0$. Then, since σ' is ground and $\mathcal{D}\mathsf{om}(\sigma') = \mathcal{V}\mathsf{ar}(l)\setminus\mathcal{V}\mathsf{ar}(r)$ by construction, the claim follows straightforwardly. If the last step has the form $\langle s_0, \pi_0\rangle \leftharpoondown_{\mathcal{R}} \langle t, \pi'\rangle$, then the claim follows trivially since each step with $\leftharpoondown_{\mathcal{R}}$ only removes trace terms from π_0. □

Hence, since any pair with an empty trace is safe the following result, which states that every pair that is *reachable* from an initial pair with an empty trace is safe, straightforwardly follows from Lemma 7:

Proposition 8. *Let $\mathcal{R}$ be a TRS. If $\langle s, [\,]\rangle \rightleftharpoons_{\mathcal{R}}^{*} \langle t, \pi\rangle$, then $\langle t, \pi\rangle$ is safe.*

Now, we state the reversibility of $\rightharpoonup_{\mathcal{R}}$, i.e., the fact that $(\rightharpoonup_{\mathcal{R}})^{-1} = \leftharpoondown_{\mathcal{R}}$ (and thus the reversibility of $\leftharpoondown_{\mathcal{R}}$ and $\rightleftharpoons_{\mathcal{R}}$, too).

Theorem 9. *Let $\mathcal{R}$ be a TRS. Given the safe pairs $\langle s, \pi\rangle$ and $\langle t, \pi'\rangle$, for all $n \geq 0$, $\langle s, \pi\rangle \rightharpoonup_{\mathcal{R}}^{n} \langle t, \pi'\rangle$ iff $\langle t, \pi'\rangle \leftharpoondown_{\mathcal{R}}^{n} \langle s, \pi\rangle$.*

Proof. ($\Rightarrow$) We prove the claim by induction on the length n of the derivation $\langle s, \pi\rangle \rightharpoonup_{\mathcal{R}}^{n} \langle t, \pi'\rangle$. Since the base case $n = 0$ is trivial, let us consider the inductive case $n > 0$. Consider a derivation $\langle s, \pi\rangle \rightharpoonup_{\mathcal{R}}^{n-1} \langle s_0, \pi_0\rangle \rightharpoonup_{\mathcal{R}} \langle t, \pi'\rangle$. By Lemma 7, both $\langle s_0, \pi_0\rangle$ and $\langle t, \pi'\rangle$ are safe. By the induction hypothesis, we have $\langle s_0, \pi_0\rangle \leftharpoondown_{\mathcal{R}}^{n-1} \langle s, \pi\rangle$. Consider now the step $\langle s_0, \pi_0\rangle \rightharpoonup_{\mathcal{R}} \langle t, \pi'\rangle$. Then, there is a position $p \in \mathcal{P}\mathsf{os}(s_0)$, a rule $\beta : l \to r \in \mathcal{R}$ and a ground substitution σ such that $s_0|_p = l\sigma$, $t = s_0[r\sigma]_p$, $\sigma' = \sigma\restriction_{\mathcal{V}\mathsf{ar}(l)\setminus\mathcal{V}\mathsf{ar}(r)}$, and $\pi' = \beta(p, \sigma') : \pi_0$. Let $\theta = \sigma\restriction_{\mathcal{V}\mathsf{ar}(r)}$. Then, we have $\langle t, \pi'\rangle \leftharpoondown_{\mathcal{R}} \langle s'_0, \pi_0\rangle$ with $t|_p = r\theta$, $\beta : l \to r \in \mathcal{R}$ and $s'_0 = t[l\theta\sigma']_p$. Moreover, since $\sigma = \theta\sigma'$, we have $s'_0 = t[l\theta\sigma']_p = t[l\sigma]_p = s_0$, and the claim follows.

($\Leftarrow$) This direction proceeds in a similar way. We prove the claim by induction on the length n of the derivation $\langle t, \pi'\rangle \leftharpoondown_{\mathcal{R}}^{n} \langle s, \pi\rangle$. As before,

we only consider the inductive case $n > 0$. Let us consider a derivation $\langle t, \pi' \rangle \leftharpoondown_{\mathcal{R}}^{n-1} \langle s_0, \pi_0 \rangle \leftharpoondown_{\mathcal{R}} \langle s, \pi \rangle$. By Lemma 7, both $\langle s_0, \pi_0 \rangle$ and $\langle s, \pi \rangle$ are safe. By the induction hypothesis, we have $\langle s_0, \pi_0 \rangle \rightharpoonup_{\mathcal{R}}^{n-1} \langle t, \pi' \rangle$. Consider now the reduction step $\langle s_0, \pi_0 \rangle \leftharpoondown_{\mathcal{R}} \langle s, \pi \rangle$. Then, we have $\pi_0 = \beta(p, \sigma') : \pi$, $\beta : l \to r \in \mathcal{R}$, and there exists a ground substitution θ with $\mathcal{D}\mathsf{om}(\theta) = \mathcal{V}\mathsf{ar}(r)$ such that $s_0|_p = r\theta$ and $s = s_0[l\theta\sigma']_p$. Moreover, since $\langle s_0, \pi_0 \rangle$ is safe, we have that $\mathcal{D}\mathsf{om}(\sigma') = \mathcal{V}\mathsf{ar}(l) \backslash \mathcal{V}\mathsf{ar}(r)$ and, thus, $\mathcal{D}\mathsf{om}(\theta) \cap \mathcal{D}\mathsf{om}(\sigma') = \varnothing$. Let $\sigma = \theta\sigma'$. Then, since $s|_p = l\sigma$ and $\mathcal{D}\mathsf{om}(\sigma') = \mathcal{V}\mathsf{ar}(l) \backslash \mathcal{V}\mathsf{ar}(r)$, we can perform the step $\langle s, \pi \rangle \rightharpoonup_{\mathcal{R}} \langle s'_0, \beta(p, \sigma') : \pi \rangle$ with $s'_0 = s[r\sigma]_p = s[r\theta\sigma']_p = s[r\theta]_p = s_0[r\theta]_p = s_0$, and the claim follows. □

The next corollary is then immediate:

Corollary 10. *Let $\mathcal{R}$ be a TRS. Given the safe pairs $\langle s, \pi \rangle$ and $\langle t, \pi' \rangle$, for all $n \geq 0$, $\langle s, \pi \rangle \rightleftharpoons_{\mathcal{R}}^{n} \langle t, \pi' \rangle$ iff $\langle t, \pi' \rangle \rightleftharpoons_{\mathcal{R}}^{n} \langle s, \pi \rangle$.*

A key issue of our notion of reversible rewriting is that the backward rewrite relation $\leftharpoondown_{\mathcal{R}}$ is deterministic (thus confluent), terminating, and has a constructive definition:

Theorem 11. *Let $\mathcal{R}$ be a TRS. Given a safe pair $\langle t, \pi' \rangle$, there exists at most one pair $\langle s, \pi \rangle$ such that $\langle t, \pi' \rangle \leftharpoondown_{\mathcal{R}} \langle s, \pi \rangle$.*

Proof. First, if there is no step using $\leftharpoondown_{\mathcal{R}}$ from $\langle t, \pi' \rangle$, the claim follows trivially. Now, assume there is at least one step $\langle t, \pi' \rangle \leftharpoondown_{\mathcal{R}} \langle s, \pi \rangle$. We prove that this is the only possible step. By definition, we have $\pi' = \beta(p, \sigma') : \pi$, $p \in \mathcal{P}\mathsf{os}(t)$, $\beta : l \to r \in \mathcal{R}$, and there exists a ground substitution θ with $\mathcal{D}\mathsf{om}(\theta) = \mathcal{V}\mathsf{ar}(r)$ such that $t|_p = r\theta$ and $s = t[l\theta\sigma']_p$. The only source of nondeterminism may come from choosing a rule labeled with β and from the computation of the substitution θ. The claim follows trivially from the fact that labels are unique in $\mathcal{R}$ and that, if there is some ground substitution θ' with $\theta' = \mathcal{V}\mathsf{ar}(r)$ and $t|_p = r\theta'$, then $\theta = \theta'$. □

Therefore, $\leftharpoondown_{\mathcal{R}}$ is clearly deterministic and confluent. Termination holds straightforwardly for pairs with finite traces since its length strictly decreases with every backward step. Note however that even when $\rightharpoonup_{\mathcal{R}}$ and $\leftharpoondown_{\mathcal{R}}$ are terminating, the relation $\rightleftharpoons_{\mathcal{R}}$ is always non-terminating since one can keep going back and forth.

3.2. Conditional Term Rewrite Systems

In this section, we extend the previous notions and results to DCTRSs. We note that considering DCTRSs is not enough to make conditional rewriting deterministic. In general, given a rewrite step $s \to_{p,\beta} t$ with p a position of s, $\beta : l \to r \Leftarrow \overline{s_n \to t_n}$ a rule, and σ a substitution such that $s|_p = l\sigma$ and $s_i\sigma \to_{\mathcal{R}}^* t_i\sigma$ for all $i = 1, \ldots, n$, there are three potential sources of non-determinism: the selected position p, the selected rule β, and the substitution σ. The use of DCTRSs can only make deterministic the last one, but the choice of a position and the selection of a rule may still be non-deterministic.

For DCTRSs, the notion of a trace term used for TRSs is not sufficient since we also need to store the traces of the subderivations associated to the condition of the applied rule (if any). Therefore, we generalize the notion of a trace as follows:

Definition 12 (trace). Given a CTRS $\mathcal{R}$, a *trace* in $\mathcal{R}$ is recursively defined as follows:

- the empty list is a trace;
- if $\pi, \pi_1, \ldots, \pi_n$ are traces in $\mathcal{R}$, $n \geq 0$, $\beta : l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n} \in \mathcal{R}$ is a rule, p is a position, and σ is a ground substitution, then $\beta(p, \sigma, \pi_1, \ldots, \pi_n) : \pi$ is a trace in $\mathcal{R}$.

We refer to each component $\beta(p, \sigma, \pi_1, \ldots, \pi_n)$ in a trace as a *trace term*.

Intuitively speaking, a trace term $\beta(p, \sigma, \pi_1, \ldots, \pi_n)$ stores the position of a reduction step, a substitution with the bindings that are required for the step to be reversible (e.g., the bindings for the erased variables, but not only; see below) and the traces associated to the subcomputations in the condition.

The notion of a safe pair is now more involved in order to deal with conditional rules. The motivation for this definition will be explained below, after introducing reversible rewriting for DCTRSs.

Definition 13 (safe pair). Let $\mathcal{R}$ be a DCTRS. A trace π is *safe* in $\mathcal{R}$ iff, for all trace terms $\beta(p, \sigma, \overline{\pi_n})$ in π, σ is a ground substitution with $\mathcal{D}\mathsf{om}(\sigma) = (\mathcal{V}\mathsf{ar}(l) \backslash \mathcal{V}\mathsf{ar}(r, \overline{s_n}, \overline{t_n})) \cup \bigcup_{i=1}^{n} \mathcal{V}\mathsf{ar}(t_i) \backslash \mathcal{V}\mathsf{ar}(r, \overline{s_{i+1,n}})$, $\beta : l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n} \in \mathcal{R}$, and $\overline{\pi_n}$ are safe too. The pair $\langle s, \pi \rangle$ is *safe* in $\mathcal{R}$ iff π is safe.

Reversible (conditional) rewriting can now be introduced as follows:

Definition 14 (reversible rewriting). Let $\mathcal{R}$ be a DCTRS. The reversible rewrite relation $\rightharpoonup_{\mathcal{R}}$ is defined on safe pairs $\langle t, \pi \rangle$, where t is a ground term

and π is a trace in $\mathcal{R}$. The reversible rewrite relation extends standard conditional rewriting as follows:

$$\langle s, \pi\rangle \rightharpoonup_{\mathcal{R}} \langle t, \beta(p, \sigma', \pi_1, \ldots, \pi_n) : \pi\rangle$$

iff there exist a position $p \in \mathcal{P}\mathsf{os}(s)$, a rewrite rule $\beta : l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n} \in \mathcal{R}$, and a ground substitution σ such that $s|_p = l\sigma$, $\langle s_i\sigma, [\,]\rangle \rightharpoonup^*_{\mathcal{R}} \langle t_i\sigma, \pi_i\rangle$ for all $i = 1, \ldots, n$, $t = s[r\sigma]_p$, and $\sigma' = \sigma\restriction_{(\mathcal{V}\mathsf{ar}(l)\setminus\mathcal{V}\mathsf{ar}(r,\overline{s_n},\overline{t_n}))\cup\bigcup_{i=1}^{n}\mathcal{V}\mathsf{ar}(t_i)\setminus\mathcal{V}\mathsf{ar}(r,\overline{s_{i+1,n}})}$. The reverse relation, $\leftharpoondown_{\mathcal{R}}$, is then defined as follows:

$$\langle t, \beta(p, \sigma', \pi_1, \ldots, \pi_n) : \pi\rangle \leftharpoondown_{\mathcal{R}} \langle s, \pi\rangle$$

iff $\langle t, \beta(p, \sigma', \overline{\pi_n}) : \pi\rangle$ is a safe pair in $\mathcal{R}$, $\beta : l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n} \in \mathcal{R}$ and there is a ground substitution θ such that $\mathcal{D}\mathsf{om}(\theta) = \mathcal{V}\mathsf{ar}(r, \overline{s_n})\setminus\mathcal{D}\mathsf{om}(\sigma')$, $t|_p = r\theta$, $\langle t_i\theta\sigma', \pi_i\rangle \leftharpoondown^*_{\mathcal{R}} \langle s_i\theta\sigma', [\,]\rangle$ for all $i = 1, \ldots, n$, and $s = t[l\theta\sigma']_p$. Note that $\theta\sigma' = \sigma'\theta = \theta \cup \sigma'$ since $\mathcal{D}\mathsf{om}(\theta) \cap \mathcal{D}\mathsf{om}(\sigma') = \varnothing$ and both substitutions are ground.

As in the unconditional case, we denote the union of both relations $\rightharpoonup_{\mathcal{R}} \cup \leftharpoondown_{\mathcal{R}}$ by $\rightleftharpoons_{\mathcal{R}}$.

Example 15. Consider again the DCTRS $\mathcal{R}$ from Example 1:

$$\begin{array}{ll}
\beta_1 : \quad \mathsf{add}(\mathsf{0}, y) \to y & \beta_4 : \quad \mathsf{even}(\mathsf{0}) \to \mathsf{true} \\
\beta_2 : \mathsf{add}(\mathsf{s}(x), y) \to \mathsf{s}(\mathsf{add}(x, y)) & \beta_5 : \mathsf{even}(\mathsf{s}(\mathsf{s}(x))) \to \mathsf{even}(x) \\
\beta_3 : \quad \mathsf{double}(x) \to \mathsf{add}(x, x) \Leftarrow \mathsf{even}(x) \twoheadrightarrow \mathsf{true} &
\end{array}$$

Given the term $\mathsf{double}(\mathsf{s}(\mathsf{s}(\mathsf{0})))$, we have, for instance, the following forward derivation:

$$\begin{array}{ll}
\langle \mathsf{double}(\mathsf{s}(\mathsf{s}(\mathsf{0}))), [\,]\rangle & \\
\quad \rightharpoonup_{\mathcal{R}} & \langle \mathsf{add}(\mathsf{s}(\mathsf{s}(\mathsf{0})), \mathsf{s}(\mathsf{s}(\mathsf{0}))), [\beta_3(\epsilon, id, \pi)]\rangle \\
\quad \rightharpoonup_{\mathcal{R}} & \cdots \\
\quad \rightharpoonup_{\mathcal{R}} & \langle \mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{0}))))), [\beta_1(1.1, id), \beta_2(1, id), \beta_2(\epsilon, id), \beta_3(\epsilon, id, \pi)]\rangle
\end{array}$$

where $\pi = [\beta_4(\epsilon, id), \beta_5(\epsilon, id)]$ since we have the following reduction:

$$\langle \mathsf{even}(\mathsf{s}(\mathsf{s}(\mathsf{0}))), [\,]\rangle \rightharpoonup_{\mathcal{R}} \langle \mathsf{even}(\mathsf{0}), [\beta_5(\epsilon, id)]\rangle \rightharpoonup_{\mathcal{R}} \langle \mathsf{true}, [\beta_4(\epsilon, id), \beta_5(\epsilon, id)]\rangle$$

The reader can easily construct the associated backward derivation:

$$\langle \mathsf{add}(\mathsf{s}(\mathsf{s}(\mathsf{0})), \mathsf{s}(\mathsf{s}(\mathsf{0}))), [\beta_1(1.1, id), \beta_2(1, id), \ldots]\rangle \leftharpoondown^*_{\mathcal{R}} \langle \mathsf{double}(\mathsf{s}(\mathsf{s}(\mathsf{0}))), [\,]\rangle$$

Let us now explain why we need to store σ' in a step of the form $\langle s, \pi\rangle \rightharpoonup_{\mathcal{R}} \langle t, \beta(p, \sigma', \overline{\pi_n}) : \pi\rangle$. Given a DCTRS, for each rule $l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n}$, the following conditions hold:

- 3-CTRS: $\mathcal{V}\mathsf{ar}(r) \subseteq \mathcal{V}\mathsf{ar}(l, \overline{s_n}, \overline{t_n})$.
- Determinism: for all $i = 1, \ldots, n$, we have $\mathcal{V}\mathsf{ar}(s_i) \subseteq \mathcal{V}\mathsf{ar}(l, \overline{t_{i-1}})$.

Intuitively, the backward relation $\leftharpoondown_{\mathcal{R}}$ can be seen as equivalent to the forward relation $\rightharpoonup_{\mathcal{R}}$ but using a reverse rule of the form $r \to l \Leftarrow t_n \twoheadrightarrow s_n, \ldots, t_1 \twoheadrightarrow s_1$. Therefore, in order to ensure that backward reduction is deterministic, we need the same conditions as above but on the reverse rule:[4]

- 3-CTRS: $\mathcal{V}\mathsf{ar}(l) \subseteq \mathcal{V}\mathsf{ar}(r, \overline{s_n}, \overline{t_n})$.
- Determinism: for all $i = 1, \ldots, n$, $\mathcal{V}\mathsf{ar}(t_i) \subseteq \mathcal{V}\mathsf{ar}(r, \overline{s_{i+1,n}})$.

Since these conditions cannot be guaranteed in general, we store

$$\sigma' = \sigma\restriction_{(\mathcal{V}\mathsf{ar}(l)\setminus\mathcal{V}\mathsf{ar}(r,\overline{s_n},\overline{t_n}))\cup\bigcup_{i=1}^{n}\mathcal{V}\mathsf{ar}(t_i)\setminus\mathcal{V}\mathsf{ar}(r,\overline{s_{i+1,n}})}$$

in the trace term so that $(r \to l \Leftarrow t_n \twoheadrightarrow s_n, \ldots, t_1 \twoheadrightarrow s_1)\sigma'$ is deterministic and fulfills the conditions of a 3-CTRS by construction, i.e., $\mathcal{V}\mathsf{ar}(l\sigma') \subseteq \mathcal{V}\mathsf{ar}(r\sigma', \overline{s_n\sigma'}, \overline{t_n\sigma'})$ and for all $i = 1, \ldots, n$, $\mathcal{V}\mathsf{ar}(t_i\sigma') \subseteq \mathcal{V}\mathsf{ar}(r\sigma', \overline{s_{i+1,n}\sigma'})$; see the proof of Theorem 21 for more details.

Example 16. Consider the following DCTRS:

$$\beta_1 : \mathsf{f}(x, y, m) \to \mathsf{s}(w) \Leftarrow \mathsf{h}(x) \twoheadrightarrow x, \mathsf{g}(y, \mathsf{4}) \twoheadrightarrow w$$
$$\beta_2 : \mathsf{h}(\mathsf{0}) \to \mathsf{0} \qquad \beta_3 : \mathsf{h}(\mathsf{1}) \to \mathsf{1} \qquad \beta_4 : \mathsf{g}(x, y) \to x$$

and the step $\langle \mathsf{f}(\mathsf{0}, \mathsf{2}, \mathsf{4}), [\,] \rangle \rightharpoonup_{\mathcal{R}} \langle \mathsf{s}(\mathsf{2}), [\beta_1(\epsilon, \sigma', \pi_1, \pi_2)] \rangle$ with $\sigma' = \{m \mapsto \mathsf{4}, x \mapsto \mathsf{0}\}$, $\pi_1 = [\beta_2(\epsilon, id)]$ and $\pi_2 = [\beta_4(\epsilon, \{y \mapsto \mathsf{4}\})]$. The binding of variable m is required to recover the value of the *erased* variable m, but the binding of variable x is also needed to perform the subderivation $\langle x, \pi_1 \rangle \leftharpoondown_{\mathcal{R}} \langle \mathsf{h}(x), [\,] \rangle$ when applying a backward step from $\langle \mathsf{s}(\mathsf{2}), [\beta_1(\epsilon, \sigma', \pi_1, \pi_2)] \rangle$. If the binding for x were unknown, this step would not be deterministic. As mentioned above, an instantiated reverse rule $(\mathsf{s}(w) \to \mathsf{f}(x, y, m) \Leftarrow w \twoheadrightarrow \mathsf{g}(y, \mathsf{4}), x \twoheadrightarrow \mathsf{h}(x))\sigma' = \mathsf{s}(w) \to \mathsf{f}(\mathsf{0}, y, \mathsf{4}) \Leftarrow w \twoheadrightarrow \mathsf{g}(y, \mathsf{4}), \mathsf{0} \twoheadrightarrow \mathsf{h}(\mathsf{0})$ would be a legal DCTRS rule thanks to σ'.

We note that similar conditions could be defined for arbitrary 3-CTRSs. However, the conditions would be much more involved; e.g., one had to compute first the *variable dependencies* between the equations in the conditions.

[4]We note that the notion of a non-erasing rule is extended to the DCTRSs in [32], which results in a similar condition.

Therefore, we prefer to keep the simpler conditions for DCTRSs (where these dependencies are fixed), which is still quite a general class of CTRSs.

Reversible rewriting is also a conservative extension of rewriting for DCTRSs (we omit the proof since it is straightforward):

Theorem 17. *Let $\mathcal{R}$ be a DCTRS. Given ground terms s, t, if $s \to_{\mathcal{R}}^* t$, then for any trace π there exists a trace π' such that $\langle s, \pi\rangle \rightharpoonup_{\mathcal{R}}^* \langle t, \pi'\rangle$.*

For the following result, we need some preliminary notions (see, e.g., [36]). For every oriented CTRS $\mathcal{R}$, we inductively define the TRSs $\mathcal{R}_k$, $k \geq 0$, as follows:

$$\begin{aligned}\mathcal{R}_0 &= \varnothing\\ \mathcal{R}_{k+1} &= \{l\sigma \to r\sigma \mid l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n} \in \mathcal{R},\ s_i\sigma \to_{\mathcal{R}_k}^* t_i\sigma \text{ for all } i = 1, \ldots, n\}\end{aligned}$$

Observe that $\mathcal{R}_k \subseteq \mathcal{R}_{k+1}$ for all $k \geq 0$. We have $\to_{\mathcal{R}} = \bigcup_{i\geq 0} \to_{\mathcal{R}_i}$. We also have $s \to_{\mathcal{R}} t$ iff $s \to_{\mathcal{R}_k} t$ for some $k \geq 0$. The minimum such k is called the *depth* of $s \to_{\mathcal{R}} t$, and the maximum depth k of $s = s_0 \to_{\mathcal{R}_{k_1}} \cdots \to_{\mathcal{R}_{k_m}} s_m = t$ (i.e., k is the maximum of depths $k_1, \ldots, k_m$) is called the *depth* of the derivation. If a derivation has depth k and length m, we write $s \to_{\mathcal{R}_k}^m t$. Analogous notions can naturally be defined for $\rightharpoonup_{\mathcal{R}}$, $\leftharpoondown_{\mathcal{R}}$, and $\rightleftharpoons_{\mathcal{R}}$.

The next result shows that safe pairs are also preserved through reversible rewriting with DCTRSs:

Lemma 18. *Let $\mathcal{R}$ be a DCTRS and $\langle s, \pi\rangle$ a safe pair. If $\langle s, \pi\rangle \rightleftharpoons_{\mathcal{R}}^* \langle t, \pi'\rangle$, then $\langle t, \pi'\rangle$ is also safe.*

Proof. We prove the claim by induction on the lexicographic product (k, m) of the depth k and the length m of the derivation $\langle s, \pi\rangle \rightleftharpoons_{\mathcal{R}_k}^m \langle t, \pi'\rangle$. Since the base case is trivial, we consider the inductive case $(k, m) > (0, 0)$. Consider a derivation $\langle s, \pi\rangle \rightleftharpoons_{\mathcal{R}_k}^{m-1} \langle s_0, \pi_0\rangle \rightleftharpoons_{\mathcal{R}_k} \langle t, \pi'\rangle$. By the induction hypothesis, we have that $\langle s_0, \pi_0\rangle$ is safe. Now, we distinguish two cases depending on the last step. If the last step is $\langle s_0, \pi_0\rangle \rightharpoonup_{\mathcal{R}_k} \langle t, \pi'\rangle$, then there exist a position $p \in \mathcal{P}\mathsf{os}(s_0)$, a rewrite rule $\beta : l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n} \in \mathcal{R}$, and a ground substitution σ such that $s_0|_p = l\sigma$, $\langle s_i\sigma, [\,]\rangle \rightharpoonup_{\mathcal{R}_{k_i}}^* \langle t_i\sigma, \pi_i\rangle$ for all $i = 1, \ldots, n$, $t = s_0[r\sigma]_p$, $\sigma' = \sigma\restriction_{(\mathcal{V}\mathsf{ar}(l)\setminus\mathcal{V}\mathsf{ar}(r,\overline{s_n},\overline{t_n}))\cup\bigcup_{i=1}^n \mathcal{V}\mathsf{ar}(t_i)\setminus\mathcal{V}\mathsf{ar}(r,\overline{s_{i+1,n}})}$, and $\pi' = \beta(p, \sigma', \pi_1, \ldots, \pi_n)$. Then, since $k_i < k$, $i = 1, \ldots, n$, σ' is ground and $\mathcal{D}\mathsf{om}(\sigma') = (\mathcal{V}\mathsf{ar}(l)\setminus\mathcal{V}\mathsf{ar}(r, \overline{s_n}, \overline{t_n})) \cup \bigcup_{i=1}^n \mathcal{V}\mathsf{ar}(t_i)\setminus\mathcal{V}\mathsf{ar}(r, \overline{s_{i+1,n}})$ by construction, the claim follows by induction. Finally, if the last step has the form $\langle s_0, \pi_0\rangle \leftharpoondown_{\mathcal{R}_k} \langle t, \pi'\rangle$, then the claim follows trivially since a step with $\leftharpoondown_{\mathcal{R}}$ only removes trace terms from π_0. □

As in the unconditional case, the following proposition follows straightforwardly from the previous lemma since any pair with an empty trace is safe.

Proposition 19. *Let $\mathcal{R}$ be a DCTRS. If $\langle s, [\,]\rangle \rightleftharpoons^*_{\mathcal{R}} \langle t, \pi\rangle$, then $\langle t, \pi\rangle$ is safe in $\mathcal{R}$.*

Now, we can already state the reversibility of $\rightharpoonup_{\mathcal{R}}$ for DCTRSs:

Theorem 20. *Let $\mathcal{R}$ be a DCTRS. Given the safe pairs $\langle s, \pi\rangle$ and $\langle t, \pi'\rangle$, for all $k, m \geq 0$, $\langle s, \pi\rangle \rightharpoonup^m_{\mathcal{R}_k} \langle t, \pi'\rangle$ iff $\langle t, \pi'\rangle \leftharpoondown^m_{\mathcal{R}_k} \langle s, \pi\rangle$.*

Proof. ($\Rightarrow$) We prove the claim by induction on the lexicographic product (k, m) of the depth k and the length m of the derivation $\langle s, \pi\rangle \rightharpoonup^m_{\mathcal{R}_k} \langle t, \pi'\rangle$. Since the base case is trivial, we consider the inductive case $(k, m) > (0, 0)$. Consider a derivation $\langle s, \pi\rangle \rightharpoonup^{m-1}_{\mathcal{R}_k} \langle s_0, \pi_0\rangle \rightharpoonup_{\mathcal{R}_k} \langle t, \pi'\rangle$ whose associated product is (k, m). By Proposition 19, both $\langle s_0, \pi_0\rangle$ and $\langle t, \pi'\rangle$ are safe. By the induction hypothesis, since $(k, m-1) < (k, m)$, we have $\langle s_0, \pi_0\rangle \leftharpoondown^{m-1}_{\mathcal{R}_k} \langle s, \pi\rangle$. Consider now the step $\langle s_0, \pi_0\rangle \rightharpoonup_{\mathcal{R}_k} \langle t, \pi'\rangle$. Thus, there exist a position $p \in \mathcal{P}\mathsf{os}(s_0)$, a rule $\beta : l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n} \in \mathcal{R}$, and a ground substitution σ such that $s_0|_p = l\sigma$, $\langle s_i\sigma, [\,]\rangle \rightharpoonup^*_{\mathcal{R}_{k_i}} \langle t_i\sigma, \pi_i\rangle$ for all $i = 1, \ldots, n$, $t = s_0[r\sigma]_p$, $\sigma' = \sigma\restriction_{(\mathcal{V}\mathsf{ar}(l)\setminus\mathcal{V}\mathsf{ar}(r,\overline{s_n},\overline{t_n}))\cup\bigcup_{i=1}^n \mathcal{V}\mathsf{ar}(t_i)\setminus\mathcal{V}\mathsf{ar}(r,\overline{s_{i+1,n}})}$, and $\pi' = \beta(p, \sigma', \pi_1, \ldots, \pi_n) : \pi_0$. By definition of $\rightharpoonup_{\mathcal{R}_k}$, we have that $k_i < k$ and, thus, $(k_i, m_1) < (k, m_2)$ for all $i = 1, \ldots, n$ and for all m_1, m_2. Hence, by the induction hypothesis, we have $\langle t_i\sigma, \pi_i\rangle \leftharpoondown^*_{\mathcal{R}_{k_i}} \langle s_i\sigma, [\,]\rangle$ for all $i = 1, \ldots, n$. Let $\theta = \sigma\restriction_{\mathcal{V}\mathsf{ar}(r,\overline{s_n})\setminus\mathcal{D}\mathsf{om}(\sigma')}$, so that $\sigma = \theta\sigma'$ and $\mathcal{D}\mathsf{om}(\theta) \cap \mathcal{D}\mathsf{om}(\sigma') = \varnothing$. Therefore, we have $\langle t, \pi'\rangle \leftharpoondown_{\mathcal{R}_k} \langle s'_0, \pi_0\rangle$ with $t|_p = r\theta$, $\beta : l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n} \in \mathcal{R}$ and $s'_0 = t[l\theta\sigma']_p = t[l\sigma]_p = s_0$, and the claim follows.

($\Leftarrow$) This direction proceeds in a similar way. We prove the claim by induction on the lexicographic product (k, m) of the depth k and the length m of the considered derivation. Since the base case is trivial, let us consider the inductive case $(k, m) > (0, 0)$. Consider a derivation $\langle t, \pi'\rangle \leftharpoondown^{m-1}_{\mathcal{R}_k} \langle s_0, \pi_0\rangle \leftharpoondown_{\mathcal{R}_k} \langle s, \pi\rangle$ whose associated product is (k, m). By Proposition 19, both $\langle s_0, \pi_0\rangle$ and $\langle s, \pi\rangle$ are safe. By the induction hypothesis, since $(k, m-1) < (k, m)$, we have $\langle s_0, \pi_0\rangle \rightharpoonup^{m-1}_{\mathcal{R}_k} \langle t, \pi'\rangle$. Consider now the step $\langle s_0, \pi_0\rangle \leftharpoondown_{\mathcal{R}_k} \langle s, \pi\rangle$. Then, we have $\pi_0 = \beta(p, \sigma', \pi_1, \ldots, \pi_n) : \pi$, $\beta : l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n} \in \mathcal{R}$, and there exists a ground substitution θ with $\mathcal{D}\mathsf{om}(\theta) = \mathcal{V}\mathsf{ar}(r, \overline{s_n})\setminus\mathcal{D}\mathsf{om}(\sigma')$ such that $s_0|_p = r\theta$, $\langle t_i\theta\sigma', \pi_i\rangle \leftharpoondown^*_{\mathcal{R}_{k_i}} \langle s_i\theta\sigma', [\,]\rangle$ for all $i = 1, \ldots, n$, and $s = s_0[l\theta\sigma']_p$. Moreover, since $\langle s_0, \pi_0\rangle$ is safe, we have that $\mathcal{D}\mathsf{om}(\sigma') = (\mathcal{V}\mathsf{ar}(l)\setminus\mathcal{V}\mathsf{ar}(r, \overline{s_n}, \overline{t_n})) \cup \bigcup_{i=1}^n \mathcal{V}\mathsf{ar}(t_i)\setminus\mathcal{V}\mathsf{ar}(r, \overline{s_{i+1,n}})$. By definition of $\leftharpoondown_{\mathcal{R}_k}$, we have that $k_i < k$ and, thus, $(k_i, m_1) < (k, m_2)$ for all $i = 1, \ldots, n$ and for all m_1, m_2. By the induction hypothesis, we have $\langle s_i\theta\sigma', [\,]\rangle \rightharpoonup^*_{\mathcal{R}_{k_i}} \langle t_i\theta\sigma', \pi_i\rangle$

for all $i = 1, \ldots, n$. Let $\sigma = \theta\sigma'$, with $\mathcal{D}\mathsf{om}(\theta) \cap \mathcal{D}\mathsf{om}(\sigma') = \varnothing$. Then, since $s|_p = l\sigma$, we can perform the step $\langle s, \pi\rangle \rightharpoonup_{\mathcal{R}_k} \langle s'_0, \beta(p, \sigma', \pi_1, \ldots, \pi_n) : \pi\rangle$ with $s'_0 = s[r\sigma]_p = s[r\theta\sigma']_p$; moreover, $s[r\theta\sigma']_p = s[r\theta]_p = s_0[r\theta]_p = s_0$ since $\mathcal{D}\mathsf{om}(\sigma') \cap \mathcal{V}\mathsf{ar}(r) = \varnothing$, which concludes the proof. □

In the following, we say that $\langle t, \pi'\rangle \leftharpoondown_{\mathcal{R}} \langle s, \pi\rangle$ is a *deterministic* step if there is no other, different pair $\langle s'', \pi''\rangle$ with $\langle t, \pi'\rangle \leftharpoondown_{\mathcal{R}} \langle s'', \pi''\rangle$ and, moreover, the subderivations for the equations in the condition of the applied rule (if any) are deterministic, too. We say that a derivation $\langle t, \pi'\rangle \leftharpoondown^*_{\mathcal{R}} \langle s, \pi\rangle$ is deterministic if each reduction step in the derivation is deterministic.

Now, we can already prove that backward reversible rewriting is also deterministic, as in the unconditional case:

Theorem 21. *Let $\mathcal{R}$ be a DCTRS. Let $\langle t, \pi'\rangle$ be a safe pair with $\langle t, \pi'\rangle \leftharpoondown^*_{\mathcal{R}} \langle s, \pi\rangle$ for some term s and trace π. Then $\langle t, \pi'\rangle \leftharpoondown^*_{\mathcal{R}} \langle s, \pi\rangle$ is deterministic.*

Proof. We prove the claim by induction on the lexicographic product (k, m) of the depth k and the length m of the steps. The case $m = 0$ is trivial, and thus we let $m > 0$. Assume $\langle t, \pi'\rangle \leftharpoondown^{m-1}_{\mathcal{R}_k} \langle u, \pi''\rangle \leftharpoondown_{\mathcal{R}_k} \langle s, \pi\rangle$. For the base case $k = 1$, the applied rule is unconditional and the proof is analogous to that of Theorem 11.

Let us now consider $k > 1$. By definition, if $\langle u, \pi''\rangle \leftharpoondown_{\mathcal{R}_k} \langle s, \pi\rangle$, we have $\pi'' = \beta(p, \sigma', \pi_1, \ldots, \pi_n) : \pi$, $\beta : l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n} \in \mathcal{R}$ and there exists a ground substitution θ with $\mathcal{D}\mathsf{om}(\theta) = \mathcal{V}\mathsf{ar}(r)$ such that $u|_p = r\theta$, $\langle t_i\theta\sigma', \pi_i\rangle \leftharpoondown^*_{\mathcal{R}_j} \langle s_i\theta\sigma', [\,]\rangle$, $j < k$, for all $i = 1, \ldots, n$, and $s = t[l\theta\sigma']_p$. By the induction hypothesis, the subderivations $\langle t_i\theta\sigma', \pi_i\rangle \leftharpoondown^*_{\mathcal{R}_j} \langle s_i\theta\sigma', [\,]\rangle$ are deterministic, i.e., $\langle s_i\theta\sigma', [\,]\rangle$ is a unique resulting term obtained by reducing $\langle t_i\theta\sigma', \pi_i\rangle$. Therefore, the only remaining source of nondeterminism can come from choosing a rule labeled with β and from the computed substitution θ. On the one hand, the labels are unique in $\mathcal{R}$. As for θ, we prove that this is indeed the only possible substitution for the reduction step. Consider the instance of rule $l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n}$ with σ': $l\sigma' \to r\sigma' \Leftarrow \overline{s_n\sigma' \twoheadrightarrow t_n\sigma'}$. Since $\langle u, \pi''\rangle$ is safe, we have that σ' is a ground substitution and $\mathcal{D}\mathsf{om}(\sigma') = (\mathcal{V}\mathsf{ar}(l) \backslash \mathcal{V}\mathsf{ar}(r, \overline{s_n}, \overline{t_n})) \cup \bigcup_{i=1}^n \mathcal{V}\mathsf{ar}(t_i) \backslash \mathcal{V}\mathsf{ar}(r, \overline{s_{i+1,n}})$. Then, the following properties hold:

- $\mathcal{V}\mathsf{ar}(l\sigma') \subseteq \mathcal{V}\mathsf{ar}(r\sigma', \overline{s_n\sigma'}, \overline{t_n\sigma'})$, since σ' is ground and it covers all the variables in $\mathcal{V}\mathsf{ar}(l) \backslash \mathcal{V}\mathsf{ar}(r, \overline{s_n}, \overline{t_n})$.
- $\mathcal{V}\mathsf{ar}(t_i\sigma') \subseteq \mathcal{V}\mathsf{ar}(r\sigma', \overline{s_{i+1,n}\sigma'})$ for all $i = 1, \ldots, n$, since σ' is ground and it covers all variables in $\bigcup_{i=1}^n \mathcal{V}\mathsf{ar}(t_i) \backslash \mathcal{V}\mathsf{ar}(r, \overline{s_{i+1,n}})$.

The above properties guarantee that a rule of the form $r\sigma' \to l\sigma' \Leftarrow t_n\sigma' \twoheadrightarrow s_n\sigma', \ldots, t_1\sigma' \twoheadrightarrow s_1\sigma'$ can be seen as a rule of a DCTRS and, thus, there exists a deterministic procedure to compute θ, which completes the proof. □

Therefore, $\leftharpoondown_{\mathcal{R}}$ is deterministic and confluent. Termination is trivially guaranteed for pairs with a finite trace since the trace's length strictly decreases with every backward step.

4. Removing Positions from Traces

Once we have a feasible definition of reversible rewriting, there are two refinements that can be considered: i) reducing the size of the traces and ii) defining a *reversibilization* transformation so that standard rewriting becomes reversible in the transformed system. In this section, we consider the first problem, leaving the second one for the next section.

In principle, one could remove information from the traces by requiring certain conditions on the considered systems. For instance, requiring injective functions may help to remove rule labels from trace terms. Also, requiring *non-erasing* rules may help to remove the second component of trace terms (i.e., the substitutions). In this section, however, we deal with a more challenging topic: removing positions from traces. This is useful not only to reduce the size of the traces but it is also essential to define a reversibilization technique for DCTRSs in the next section.[5] In particular, we aim at transforming a given DCTRS into one that fulfills some conditions that make storing positions unnecessary.

In the following, given a CTRS $\mathcal{R}$, we say that a term t is *basic* [18] if it has the form $f(\overline{t_n})$ with $f \in \mathcal{D}_{\mathcal{R}}$ a defined function symbol and $\overline{t_n} \in \mathcal{T}(\mathcal{C}_{\mathcal{R}}, \mathcal{V})$ constructor terms. Furthermore, in the remainder of this paper, we assume that the right-hand sides of the equations in the conditions of the rules of a DCTRS are constructor terms. This is not a significant restriction since these terms cannot be reduced anyway (since we consider oriented equations in this paper), and still covers most practical examples.

Now, we introduce the following subclass of DCTRSs:

Definition 22 (pcDCTRS [30]). We say that a DCTRS $\mathcal{R}$ is a pcDCTRS ("pc" stands for *pure constructor*) if, for each rule $l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n} \in \mathcal{R}$, we have that l and $\overline{s_n}$ are basic terms and r and $\overline{t_n}$ are constructor terms.

[5] We note that defining a transformation with traces that include positions would be a rather difficult task because positions are *dynamic* (i.e., they depend on the term being reduced) and thus would require a complex (and inefficient) system instrumentation.

Pure constructor systems are called *normalized* systems in [3]. Also, they are mostly equivalent to the class III_n of conditional systems in [8], where $t_1, \ldots, t_n$ are required to be ground unconditional normal forms instead.[6]

In principle, any DCTRS with basic terms in the left-hand sides (i.e., a *constructor* DCTRS) and constructor terms in the right-hand sides of the equations of the rules can be transformed into a pcDCTRS by applying a few simple transformations: flattening and simplification of constructor conditions. Let us now consider each of these transformations separately. Roughly speaking, flattening involves transforming a term (occurring, e.g., in the right-hand side of a DCTRS or in the condition) with nested defined functions like $\mathsf{f}(\mathsf{g}(x))$ into a term $\mathsf{f}(y)$ and an (oriented) equation $\mathsf{g}(x) \twoheadrightarrow y$, where y is a fresh variable. Formally,

Definition 23 (flattening). Let $\mathcal{R}$ be a CTRS, $R = (l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n}) \in \mathcal{R}$ be a rule and R' be a new rule either of the form $l \to r \Leftarrow s_1 \twoheadrightarrow t_1, \ldots, s_i|_p \twoheadrightarrow w, s_i[w]_p \twoheadrightarrow t_i, \ldots, s_n \twoheadrightarrow t_n$, for some $p \in \mathcal{P}\mathsf{os}(s_i)$, $1 \leqslant i \leqslant n$, or $l \to r[w]_q \Leftarrow \overline{s_n \twoheadrightarrow t_n}, r|_q \twoheadrightarrow w$, for some $q \in \mathcal{P}\mathsf{os}(r)$, where w is a fresh variable.[7] Then, a CTRS $\mathcal{R}'$ is obtained from $\mathcal{R}$ by a *flattening* step if $\mathcal{R}' = (\mathcal{R} \backslash \{R\}) \cup \{R'\}$.

Note that, if an unconditional rule is non-erasing (i.e., $\mathcal{V}\mathsf{ar}(l) \subseteq \mathcal{V}\mathsf{ar}(r)$ for a rule $l \to r$), any conditional rule obtained by flattening is trivially non-erasing too, according to the notion of non-erasingness for DCTRSs in [32].[8]

Flattening is trivially *complete* since any flattening step can be undone by binding the fresh variable again to the selected subterm and, then, proceeding as in the original system. Soundness is more subtle though. In this work, we prove the correctness of flattening for arbitrary DCTRSs with respect to *innermost* rewriting. As usual, the innermost rewrite relation, in symbols, $\xrightarrow{\mathsf{i}}_{\mathcal{R}}$, is defined as the smallest binary relation satisfying the following: given ground terms $s, t \in \mathcal{T}(\mathcal{F})$, we have $s \xrightarrow{\mathsf{i}}_{\mathcal{R}} t$ iff there exist a position p in s such that no proper subterms of $s|_p$ are reducible, a rewrite rule $l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n} \in \mathcal{R}$, and a normalized ground substitution σ such that $s|_p = l\sigma$, $s_i\sigma \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} t_i\sigma$, for all $i = 1, \ldots, n$, and $t = s[r\sigma]_p$.

In order to prove the correctness of flattening, we state the following auxiliary lemma:

[6] Given a CTRS $\mathcal{R}$, we define $\mathcal{R}_u = \{l \to r \mid l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n} \in \mathcal{R}\}$. A term is an *unconditional* normal form in $\mathcal{R}$, if it is a normal form in $\mathcal{R}_u$.

[7] The positions p, q can be required to be different from ϵ, but this is not strictly necessary.

[8] Roughly, a DCTRS is considered non-erasing in [32] if its transformation into an unconditional TRS by an unraveling transformation gives rise to a non-erasing TRS.

Lemma 24. *Let $\mathcal{R}$ be a DCTRS. Given terms s and t, with t a normal form, and a position $p \in \mathcal{P}\mathsf{os}(s)$, we have $s \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} t$ iff $s|_p \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} w\sigma$ and $s[w\sigma]_p \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} t$, for some fresh variable w and normalized substitution σ.*

Proof. ($\Rightarrow$) Let us consider an arbitrary position $p \in \mathcal{P}\mathsf{os}(s)$. If $s|_p$ is normalized, the proof is straightforward. Otherwise, since we use innermost reduction (*leftmost* innermost, for simplicity), we can represent the derivation $s \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} t$ as follows:

$$s[s|_p]_p \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} s'[s|_p]_p \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} s'[s'']_p \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} t$$

where s'' is a normal form and the subderivation $s[s|_p]_p \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} s'[s|_p]_p$ reduces the leftmost innermost subterms that are to the left of $s|_p$ (if any). Then, by choosing $\sigma = \{w \mapsto s''\}$ we have $s|_p \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} w\sigma$ (by mimicking the steps of $s'[s|_p]_p \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} s'[s'']_p$), $s[w\sigma]_p \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} s'[w\sigma]_p$ (by mimicking the steps of $s[s|_p]_p \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} s'[s|_p]_p$), and $s'[w\sigma]_p \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} t$ (by mimicking the steps of $s'[s'']_p \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} t$), which concludes the proof.

($\Leftarrow$) This direction is perfectly analogous to the previous case. We consider an arbitrary position $p \in \mathcal{P}\mathsf{os}(s)$ such that $s|_p$ is not normalized (otherwise, the proof is trivial). Now, since derivations are innermost, we can consider that $s[w\sigma]_p \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} t$ is as follows: $s[w\sigma]_p \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} s'[w\sigma]_p \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} t$, where $s[w\sigma]_p \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} s'[w\sigma]_p$ reduces the innermost subterms to the left of $s|_p$. Therefore, we have $s[s|_p]_p \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} s'[s|_p]_p$ (by mimicking the steps of $s[w\sigma]_p \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} s'[w\sigma]_p$), $s'[s|_p]_p \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} s'[s'']_p$ (by mimicking the steps of $s|_p \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} w\sigma$, with $\sigma = \{w \mapsto s''\}$), and $s'[s'']_p \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} t$ (by mimicking the steps of $s'[w\sigma]_p \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} t$). □

The following theorem is an easy consequence of the previous lemma:

Theorem 25. *Let $\mathcal{R}$ be a DCTRS. If $\mathcal{R}'$ is obtained from $\mathcal{R}$ by a flattening step, then $\mathcal{R}'$ is a DCTRS and, for all ground terms s, t, with t a normal form, we have $s \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} t$ iff $s \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}'} t$.*

Proof. ($\Rightarrow$) We prove the claim by induction on the lexicographic product (k, m) of the depth k and the length m of the derivation $s \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}_k} t$. Since the base case is trivial, we consider the inductive case $(k, m) > (0, 0)$. Assume that $s \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}_k} t$ has the form $s[l\sigma]_u \xrightarrow{\mathsf{i}}_{\mathcal{R}_k} s[r\sigma]_u \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}_k} t$ with $l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n} \in \mathcal{R}$ and $s_i\sigma \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}_{k_i}} t_i\sigma$, $k_i < k$, $i = 1, \ldots, n$. If $l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n} \in \mathcal{R}'$, the claim follows directly by induction. Otherwise, we have that either $l \to r \Leftarrow s_1 \twoheadrightarrow$

$t_1, \ldots, s_i|_p \twoheadrightarrow w, s_i[w]_p \twoheadrightarrow t_i, \ldots, s_n \twoheadrightarrow t_n \in \mathcal{R}'$, for some $p \in \mathcal{Pos}(s_i)$, $1 \leqslant i \leqslant n$, or $l \to r[w]_q \Leftarrow \overline{s_n \twoheadrightarrow t_n}, r|_q \twoheadrightarrow w \in \mathcal{R}'$, for some $q \in \mathcal{Pos}(r)$, where w is a fresh variable. Consider first the case $l \to r \Leftarrow s_1 \twoheadrightarrow t_1, \ldots, s_i|_p \twoheadrightarrow w, s_i[w]_p \twoheadrightarrow t_i, \ldots, s_n \twoheadrightarrow t_n \in \mathcal{R}'$, for some $p \in \mathcal{Pos}(s_i)$, $1 \leqslant i \leqslant n$. Since $s_i\sigma \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}_{k_i}} t_i\sigma$, $k_i < k$, $i = 1, \ldots, n$, by the induction hypothesis, we have $s_i\sigma \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}'} t_i\sigma$, $i = 1, \ldots, n$. By Lemma 24, there exists $\sigma' = \{w \mapsto s'\}$ for some normal form s' such that $s_i|_p\sigma = s_i|_p\sigma\sigma' \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}_{k_i}} w\sigma\sigma' = w\sigma'$ and $s_i[w]_p\sigma\sigma' = s_i\sigma[w\sigma']_p \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}_{k_i}} t_i$. Moreover, since w is an extra variable, we also have $s_j\sigma\sigma' = s_j\sigma \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}'} t_j\sigma = t_j\sigma\sigma'$ for $j = 1, \ldots, i-1, i+1, \ldots, n$. Therefore, since $l\sigma\sigma' = l\sigma$ and $r\sigma\sigma' = r\sigma$, we have $s[l\sigma]_u \xrightarrow{\mathsf{i}}_{\mathcal{R}} s[r\sigma]_u$, and the claim follows by induction. Consider the second case. By the induction hypothesis, we have $s[r\sigma]_u \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}'} t$ and $s_i\sigma \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}'} t_i\sigma$ for all $i = 1, \ldots, n$. By Lemma 24, there exists a substitution $\sigma' = \{w \mapsto s'\}$ such that s' is the normal form of $r|_q\sigma$ and we have $r|_q\sigma \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}'} w\sigma'$ and $s[r\sigma[w\sigma']_q]_u \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}'} t$. Moreover, since w is a fresh variable, we have $s_i\sigma\sigma' \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}'} t_i\sigma\sigma'$ for all $i = 1, \ldots, n$. Therefore, we have $s[l\sigma\sigma']_u = s[l\sigma]_u \xrightarrow{\mathsf{i}}_{\mathcal{R}'} s[r\sigma[w\sigma']_q]_u$, which concludes the proof.

($\Leftarrow$) This direction is perfectly analogous to the previous one, and follows easily by Lemma 24 too. □

Let us now consider the second kind of transformations: the simplification of constructor conditions. Basically, we can drop an equation $s \twoheadrightarrow t$ when the terms s and t are constructor, called a *constructor condition*, by either applying the *most general unifier* (mgu) of s and t (if it exists) to the remaining part of the rule, or by deleting entirely the rule if they do not unify because (under innermost rewriting) the equation will never be satisfied by any normalized substitution. Similar transformations can be found in [33].

In order to justify these transformations, we state and prove the following results. In the following, we let $mgu(s, t)$ denote the most general unifier of terms s and t if it exists, and *fail* otherwise.

Theorem 26 (removal of unifiable constructor conditions). *Let $\mathcal{R}$ be a DCTRS and let $R = (l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n}) \in \mathcal{R}$ be a rule with $mgu(s_i, t_i) = \theta$, for some $i \in \{1, \ldots, n\}$, where s_i and t_i are constructor terms. Let R' be a new rule of the form $l\theta \to r\theta \Leftarrow s_1\theta \twoheadrightarrow t_1\theta, \ldots, s_{i-1}\theta \twoheadrightarrow t_{i-1}\theta, s_{i+1}\theta \twoheadrightarrow t_{i+1}\theta, \ldots, s_n\theta \twoheadrightarrow t_n\theta$.[9] Then $\mathcal{R}' = (\mathcal{R} \backslash \{R\}) \cup \{R'\}$ is a DCTRS and, for all*

[9] In [33], the condition $\mathcal{Dom}(\theta) \cap \mathcal{V}ar(l, r, s_1, t_1, \ldots, s_n, t_n) = \varnothing$ is required, but this condition is not really necessary.

ground terms s and t, we have $s \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} t$ iff $s \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}'} t$.

PROOF. ($\Rightarrow$) First, we prove the following claim by induction on the lexicographic product (k, m) of the depth k and the length m of the steps: if $s \xrightarrow{\mathsf{i}}{}^m_{\mathcal{R}_k} t$, then $s \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}'} t$. It suffices to consider the case where R is applied, i.e., $s = s[l\sigma]_p \xrightarrow{\mathsf{i}}_{\{R\}} s[r\sigma]_p$ with $s_j\sigma \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}_{k_j}} t_j\sigma$ for all $j \in \{1, \ldots, n\}$. By definition, σ is normalized. Hence, since s_i and t_i are constructor terms, we have that $s_i\sigma$ and $t_i\sigma$ are trivially normal forms since the normalized subterms introduced by σ cannot become reducible in a constructor context. Therefore, we have $s_i\sigma = t_i\sigma$. Thus, σ is a unifier of s_i and t_i and, hence, θ is more general than σ. Let δ be a substitution such that $\sigma = \theta\delta$. Since σ is normalized, so is δ. Since $k_j < k$ for all $j = 1, \ldots, n$, by the induction hypothesis, we have that $s_j\sigma = s_j\theta\delta \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}'} t_j\theta\delta = t_j\sigma$ for $j \in \{1, \ldots, i-1, i+1, \ldots, n\}$. Therefore, we have that $s[l\sigma]_p = s[l\theta\delta]_p \xrightarrow{\mathsf{i}}_{\{R'\}} s[r\theta\delta]_p = s[r\sigma]_p$.

($\Leftarrow$) Now, we prove the following claim by induction on the lexicographic product (k, m) of the depth k and the length m of the steps: if $s \xrightarrow{\mathsf{i}}{}^m_{\mathcal{R}'_k} t$, then $s \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} t$. It suffices to consider the case where R' is applied, i.e., $s = s[l\theta\delta]_p \xrightarrow{\mathsf{i}}_{\{R\}} s[r\theta\delta]_p$ with $s_j\theta\delta \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}'_{k_j}} t_j\theta\delta$ for all $j \in \{1, \ldots, i-1, i+1, \ldots, n\}$. By the assumption and the definition, θ and δ are normalized, and thus, $s_i\theta\delta$ and $t_i\theta\delta$ are normal forms (as in the previous case, because the normalized subterms introduced by $\theta\delta$ cannot become reducible in a constructor context), i.e., $s_i\theta\delta = t_i\theta\delta$. Since $k_j < k$ for all $j \in \{1, \ldots, i-1, i+1, \ldots, n\}$, by the induction hypothesis, we have that $s_j\theta\delta \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} t_j\theta\delta$ for $j \in \{1, \ldots, i-1, i+1, \ldots, n\}$. Therefore, we have that $s[l\sigma]_p = s[l\theta\delta]_p \xrightarrow{\mathsf{i}}_{\{R\}} s[r\theta\delta]_p = s[r\sigma]$ with $\sigma = \theta\delta$. □

Now we consider the case when the terms in the constructor condition do not unify:

Theorem 27 (removal of infeasible rules). *Let $\mathcal{R}$ be a DCTRS and let $R = (l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n}) \in \mathcal{R}$ be a rule with $mgu(s_i, t_i) = fail$, for some $i \in \{1, \ldots, n\}$. Then $\mathcal{R}' = \mathcal{R} \backslash \{R\}$ is a DCTRS and, for all ground terms s and t, we have $s \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} t$ iff $s \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}'} t$.*

PROOF. Since $\mathcal{R} \supseteq \mathcal{R}'$, the *if* part is trivial, and thus, we consider the *only-if* part. To apply R to a term, there must exist a normalized substitution σ such that $s_i\sigma \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} t_i\sigma$. Since s_i, t_i are constructor terms and σ is normalized, $s_i\sigma$ is a normal form (because the normalized subterms introduced by σ

cannot become reducible in a constructor context). If $s_i\sigma \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} t_i\sigma$ is satisfied (i.e., $s_i\sigma = t_i\sigma$), then s_i and t_i are unifiable, and thus, this contradicts the assumption. Therefore, R is never applied to any term, and hence, $s \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} t$ iff $s \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}'} t$. □

Using flattening and the simplification of constructor conditions, any constructor DCTRS with constructor terms in the right-hand sides of the equations of the rules can be transformed into a pcDCTRS. One can use, for instance, the following simple algorithm. Let $\mathcal{R}$ be such a constructor DCTRS. We apply the following transformations as much as possible:

(flattening-rhs) Assume that $\mathcal{R}$ contains a rule of the form $R = (l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n})$ where r is not a constructor term. Let $r|_q$, $q \in \mathcal{P}\mathsf{os}(r)$, be a basic subterm of r. Then, we replace rule R by a new rule of the form $l \to r[w]_q \Leftarrow \overline{s_n \twoheadrightarrow t_n}, r|_q \twoheadrightarrow w$, where w is a fresh variable.

(flattening-condition) Assume that $\mathcal{R}$ contains a rule of the form $R = (l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n})$ where s_i is neither a constructor term nor a basic term, $i \in \{1, \ldots, n\}$. Let $s_i|_q$, $q \in \mathcal{P}\mathsf{os}(s_1)$, be a basic subterm of s_i. Then, we replace rule R by a new rule of the form $l \to r \Leftarrow s_1 \twoheadrightarrow t_1, \ldots, s_i|_q \twoheadrightarrow w, s_i[w]_q \twoheadrightarrow t_i, \ldots, s_n \twoheadrightarrow t_n$, where w is a fresh variable.

(removal-unify) Assume that $\mathcal{R}$ contains a rule of the form $R = (l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n})$ where s_i is a constructor term, $i \in \{1, \ldots, n\}$. If $mgu(s_i, t_i) = \theta \neq \mathit{fail}$, then we replace rule R by a new rule of the form $l\theta \to r\theta \Leftarrow s_1\theta \twoheadrightarrow t_1\theta, \ldots, s_{i-1}\theta \twoheadrightarrow t_{i-1}\theta, s_{i+1}\theta \twoheadrightarrow t_{i+1}\theta, \ldots, s_n\theta \twoheadrightarrow t_n\theta$.

(removal-fail) Assume that $\mathcal{R}$ contains a rule of the form $R = (l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n})$ where s_i is a constructor term, $i \in \{1, \ldots, n\}$. If $mgu(s_i, t_i) = \mathit{fail}$, then we remove rule R from $\mathcal{R}$.

Trivially, by applying rule flattening-rhs as much as possible, we end up with a DCTRS where all the right-hand sides are constructor terms; analogously, the exhaustive application of rule flattening-condition allows us to ensure that the left-hand sides of all equations in the conditions of the rules are either constructor or basic; finally, the application of rules removal-unify and removal-fail produces a pcDCTRS by removing those equations in which the left-hand side is a constructor term. Therefore, in the remainder of this paper, we only consider pcDCTRSs.

A nice property of pcDCTRSs is that one can consider reductions only at *topmost* positions. Formally, given a pcDCTRS $\mathcal{R}$, we say that $s \to_{p,l\to r\Leftarrow\overline{s_n\twoheadrightarrow t_n}} t$ is a *top* reduction step if $p = \epsilon$, there is a ground substitution σ with $s = l\sigma$,

$s_i\sigma \to_{\mathcal{R}}^* t_i\sigma$ for all $i = 1, \ldots, n$, $t = r\sigma$, and all the steps in $s_i\sigma \to_{\mathcal{R}}^* t_i\sigma$ for $i = 1, \ldots, n$ are also top reduction steps. We denote top reductions with $\overset{\epsilon}{\to}$ for standard rewriting, and $\overset{\epsilon}{\underset{\mathcal{R}}{\to}}, \overset{\epsilon}{\underset{\mathcal{R}}{\leftarrow}}$ for our reversible rewrite relations.

The following result basically states that $\overset{\mathsf{i}}{\to}$ and $\overset{\epsilon}{\to}$ are equivalent for pcDCTRSs:

Theorem 28. *Let $\mathcal{R}$ be a constructor DCTRS with constructor terms in the right-hand sides of the equations and $\mathcal{R}'$ be a pcDCTRS obtained from $\mathcal{R}$ by a sequence of transformations of flattening and simplification of constructor conditions. Given ground terms s and t such that s is basic and t is normalized, we have $s \overset{\mathsf{i}}{\to}_{\mathcal{R}}^* t$ iff $s \overset{\epsilon}{\to}_{\mathcal{R}'}^* t$.*

Proof. First, it is straightforward to see that an innermost reduction in $\mathcal{R}'$ can only reduce the topmost positions of terms since defined functions can only occur at the root of terms and the terms introduced by instantiation are, by definition, irreducible. Therefore, the claim is a consequence of Theorems 25, 26 and 27, together with the above fact. □

Therefore, when considering pcDCTRSs and top reductions, storing the reduced positions in the trace terms becomes redundant since they are always ϵ. Thus, in practice, one can consider simpler trace terms without positions, $\beta(\sigma, \pi_1, \ldots, \pi_n)$, that implicitly represent the trace term $\beta(\epsilon, \sigma, \pi_1, \ldots, \pi_n)$.

Example 29. Consider the following TRS $\mathcal{R}$ defining addition and multiplication on natural numbers, and its associated pcDCTRS $\mathcal{R}'$:

$$\begin{aligned}\mathcal{R} = \{ \quad \mathsf{add}(\mathsf{0}, y) &\to y,\\ \mathsf{add}(\mathsf{s}(x), y) &\to \mathsf{s}(\mathsf{add}(x, y)),\\ \mathsf{mult}(\mathsf{0}, y) &\to \mathsf{0},\\ \mathsf{mult}(\mathsf{s}(x), y) &\to \mathsf{add}(\mathsf{mult}(x, y), y)\}\end{aligned}$$

$$\begin{aligned}\mathcal{R}' = \{ \quad \mathsf{add}(\mathsf{0}, y) &\to y,\\ \mathsf{add}(\mathsf{s}(x), y) &\to \mathsf{s}(z) \Leftarrow \mathsf{add}(x, y) \twoheadrightarrow z,\\ \mathsf{mult}(\mathsf{0}, y) &\to \mathsf{0},\\ \mathsf{mult}(\mathsf{s}(x), y) &\to w \Leftarrow \mathsf{mult}(x, y) \twoheadrightarrow z, \mathsf{add}(z, y) \twoheadrightarrow w\}\end{aligned}$$

For instance, given the following reduction in $\mathcal{R}$:

$$\mathsf{mult}(\mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{0})) \overset{\mathsf{i}}{\to}_{\mathcal{R}} \mathsf{add}(\mathsf{mult}(\mathsf{0}, \mathsf{s}(\mathsf{0})), \mathsf{s}(\mathsf{0})) \overset{\mathsf{i}}{\to}_{\mathcal{R}} \mathsf{add}(\mathsf{0}, \mathsf{s}(\mathsf{0})) \overset{\mathsf{i}}{\to}_{\mathcal{R}} \mathsf{s}(\mathsf{0})$$

we have the following counterpart in $\mathcal{R}'$:

$$\mathsf{mult}(\mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{0})) \overset{\epsilon}{\to}_{\mathcal{R}'} \mathsf{s}(\mathsf{0}) \quad \text{with } \mathsf{mult}(\mathsf{0}, \mathsf{s}(\mathsf{0})) \overset{\epsilon}{\to}_{\mathcal{R}'} \mathsf{0} \text{ and } \mathsf{add}(\mathsf{0}, \mathsf{s}(\mathsf{0})) \overset{\epsilon}{\to}_{\mathcal{R}'} \mathsf{s}(\mathsf{0})$$

Trivially, all results in Section 3 hold for pcDCTRSs and top reductions starting from basic terms. The simpler trace terms without positions will allow us to introduce appropriate injectivization and inversion transformations in the next section.

5. Reversibilization

In this section, we aim at *compiling* the reversible extension of rewriting into the system rules. Intuitively speaking, given a pure constructor system $\mathcal{R}$, we aim at producing new systems $\mathcal{R}_f$ and $\mathcal{R}_b$ such that standard rewriting in $\mathcal{R}_f$, i.e., $\to_{\mathcal{R}_f}$, coincides with the forward reversible extension $\rightharpoonup_{\mathcal{R}}$ in the original system, and analogously $\to_{\mathcal{R}_b}$ is equivalent to $\leftharpoonup_{\mathcal{R}}$. Therefore, $\mathcal{R}_f$ can be seen as an *injectivization* of $\mathcal{R}$, and $\mathcal{R}_b$ as the *inversion* of $\mathcal{R}_f$.

In principle, we could easily introduce a transformation for pcDCTRSs that mimicks the behavior of the reversible extension of rewriting. For instance, given the pcDCTRS $\mathcal{R}$ of Example 16, we could produce the following injectivized version $\mathcal{R}_f$:[10]

$$\begin{aligned}
\langle \mathsf{f}(x,y,m), ws\rangle &\to \langle \mathsf{s}(w), \beta_1(m,x,w_1,w_2) : ws\rangle \\
&\qquad \Leftarrow \langle \mathsf{h}(x), [\,]\rangle \twoheadrightarrow \langle x, w_1\rangle, \langle \mathsf{g}(y,\mathsf{4}), [\,]\rangle \twoheadrightarrow \langle w, w_2\rangle \\
\langle \mathsf{h}(\mathsf{0}), ws\rangle &\to \langle \mathsf{0}, \beta_2 : ws\rangle \\
\langle \mathsf{h}(\mathsf{1}), ws\rangle &\to \langle \mathsf{1}, \beta_3 : ws\rangle \\
\langle \mathsf{g}(x,y), ws\rangle &\to \langle x, \beta_4(y) : ws\rangle
\end{aligned}$$

For instance, the reversible step $\langle \mathsf{f}(\mathsf{0},\mathsf{2},\mathsf{4}), [\,]\rangle \xrightarrow{\epsilon}_{\mathcal{R}} \langle \mathsf{s}(\mathsf{2}), [\beta_1(\sigma', \pi_1, \pi_2)]\rangle$ with $\sigma' = \{m \mapsto \mathsf{4}, x \mapsto \mathsf{0}\}$, $\pi_1 = [\beta_2(id)]$ and $\pi_2 = [\beta_4(\{y \mapsto \mathsf{4}\})]$, has the following counterpart in $\mathcal{R}_f$:

$$\begin{aligned}
&\langle \mathsf{f}(\mathsf{0},\mathsf{2},\mathsf{4}), [\,]\rangle \xrightarrow{\epsilon}_{\mathcal{R}_f} \langle \mathsf{s}(\mathsf{2}), [\beta_1(\mathsf{4},\mathsf{0},[\beta_2],[\beta_4(\mathsf{4})])]\rangle \\
&\qquad \text{with} \quad \langle \mathsf{h}(\mathsf{0}), [\,]\rangle \xrightarrow{\epsilon}_{\mathcal{R}_f} \langle \mathsf{0}, [\beta_2]\rangle \quad \text{and} \quad \langle \mathsf{g}(\mathsf{2},\mathsf{4}), [\,]\rangle \xrightarrow{\epsilon}_{\mathcal{R}_f} \langle \mathsf{2}, [\beta_4(\mathsf{4})]\rangle
\end{aligned}$$

The only subtle difference here is that a trace term like

$$\beta_1(\{m \mapsto \mathsf{4}, x \mapsto \mathsf{0}\}, [\beta_2(id)], [\beta_4(\{y \mapsto \mathsf{4}\})])$$

is now stored in the transformed system as

$$\beta_1(\mathsf{4},\mathsf{0},[\beta_2],[\beta_4(\mathsf{4})])$$

[10] We will write just β instead of $\beta()$ when no argument is required.

Furthermore, we could produce an inverse $\mathcal{R}_b$ of the above system as follows:

$$\begin{aligned}
\langle \mathsf{s}(w), \beta_1(m,x,w_1,w_2) : ws\rangle^{-1} &\to \langle \mathsf{f}(x,y,m), ws\rangle^{-1} \\
&\Leftarrow \langle w, w_2\rangle^{-1} \twoheadrightarrow \langle \mathsf{g}(y,\mathsf{4}), [\,]\rangle^{-1}, \\
&\quad\ \langle x, w_1\rangle^{-1} \twoheadrightarrow \langle \mathsf{h}(x), [\,]\rangle^{-1} \\
\langle \mathsf{0}, \beta_2 : ws\rangle^{-1} &\to \langle \mathsf{h}(\mathsf{0}), ws\rangle^{-1} \\
\langle \mathsf{1}, \beta_3 : ws\rangle^{-1} &\to \langle \mathsf{h}(\mathsf{1}), ws\rangle^{-1} \\
\langle x, \beta_4(y) : ws\rangle^{-1} &\to \langle \mathsf{g}(x,y), ws\rangle^{-1}
\end{aligned}$$

mainly by switching the left- and right-hand sides of each rule and condition. The correctness of these injectivization and inversion transformations would be straightforward.

These transformations are only aimed at mimicking, step by step, the reversible relations $\rightharpoonup_{\mathcal{R}}$ and $\leftharpoonup_{\mathcal{R}}$. Roughly speaking, for each step $\langle s,\pi\rangle \rightharpoonup_{\mathcal{R}} \langle t,\pi'\rangle$ in a system $\mathcal{R}$, we have $\langle s,\pi\rangle \to_{\mathcal{R}_f} \langle t,\pi'\rangle$, where $\mathcal{R}_f$ is the injectivized version of $\mathcal{R}$, and for each step $\langle s,\pi\rangle \leftharpoonup_{\mathcal{R}} \langle t,\pi'\rangle$ in $\mathcal{R}$, we have $\langle s,\pi\rangle \to_{\mathcal{R}_b} \langle t,\pi'\rangle$, where $\mathcal{R}_b$ is the inverse of $\mathcal{R}_f$. More details on this approach can be found in [31]. Unfortunately, it might be much more useful to produce injective and inverse versions of *each function* defined in a system $\mathcal{R}$. Note that, in the above approach, the system $\mathcal{R}_f$ only defines a single function $\langle \_,\_\rangle$ and $\mathcal{R}_b$ only defines $\langle \_,\_\rangle^{-1}$, i.e., we are computing systems that define the relations $\rightharpoonup_{\mathcal{R}}$ and $\leftharpoonup_{\mathcal{R}}$ rather than the injectivized and inverse versions of the functions in $\mathcal{R}$. In the following, we introduce more refined transformations that can actually produce injective and inverse versions of the original functions.

5.1. Injectivization

In principle, given a function f, one can consider that the injectivization of a rule of the form[11]

$$\beta : \mathsf{f}(\overline{s_0}) \to r \Leftarrow \mathsf{f_1}(\overline{s_1}) \twoheadrightarrow t_1, \ldots, \mathsf{f_n}(\overline{s_n}) \twoheadrightarrow t_n$$

produces the following rule

$$\mathsf{f^i}(\overline{s_0}) \to \langle r, \beta(\overline{y}, \overline{w_n})\rangle \Leftarrow \mathsf{f}_1^{\mathsf{i}}(\overline{s_1}) \twoheadrightarrow \langle t_1, w_1\rangle \ldots, \mathsf{f}_n^{\mathsf{i}}(\overline{s_n}) \twoheadrightarrow \langle t_n, w_n\rangle$$

where $\{\overline{y}\} = (\mathcal{V}\mathsf{ar}(l)\backslash\mathcal{V}\mathsf{ar}(r,\overline{s_n},\overline{t_n})) \cup \bigcup_{i=1}^{n} \mathcal{V}\mathsf{ar}(t_i)\backslash\mathcal{V}\mathsf{ar}(r,\overline{s_{i+1,n}})$ and $\overline{w_n}$ are fresh variables. The following example, though, illustrates that this is not correct in general.

[11] By abuse of notation, here we let $\overline{s_0}, \ldots, \overline{s_n}$ denote sequences of terms of arbitrary length, i.e., $\overline{s_0} = s_{0,1}, \ldots, s_{0,l_0}$, $\overline{s_1} = s_{1,1}, \ldots, s_{1,l_1}$, etc.

Example 30. Consider the following pcDCTRS $\mathcal{R}$:

$$\begin{array}{lrcl} \beta_1: & \mathsf{f}(x,y) & \to & z \Leftarrow \mathsf{h}(y) \twoheadrightarrow w,\ \mathsf{first}(x,w) \twoheadrightarrow z \\ \beta_2: & \mathsf{h}(0) & \to & 0 \\ \beta_3: & \mathsf{first}(x,y) & \to & x \end{array}$$

together with the following top reduction:

$$\begin{array}{ll} \mathsf{f}(2,1) \xrightarrow{\epsilon}_{\mathcal{R}} 2 & \text{with } \sigma = \{x \mapsto 2, y \mapsto 1, w \mapsto \mathsf{h}(1), z \mapsto 2\} \\ & \text{where } \mathsf{h}(y)\sigma = \mathsf{h}(1) \xrightarrow{\epsilon}{}^*_{\mathcal{R}} \mathsf{h}(1) = w\sigma \\ & \text{and } \mathsf{first}(x,w)\sigma = \mathsf{first}(2,\mathsf{h}(1)) \xrightarrow{\epsilon}_{\mathcal{R}} 2 = z\sigma \end{array}$$

Following the scheme above, we would produce the following pcDCTRS

$$\begin{array}{rcl} \mathsf{f}^{\mathsf{i}}(x,y) & \to & \langle z, \beta_1(w_1,w_2)\rangle \Leftarrow \mathsf{h}^{\mathsf{i}}(y) \twoheadrightarrow \langle w, w_1\rangle,\ \mathsf{first}^{\mathsf{i}}(x,w) \twoheadrightarrow \langle z, w_2\rangle \\ \mathsf{h}^{\mathsf{i}}(0) & \to & \langle 0, \beta_2\rangle \\ \mathsf{first}^{\mathsf{i}}(x,y) & \to & \langle x, \beta_3(y)\rangle \end{array}$$

Unfortunately, the corresponding reduction for $\mathsf{f}^{\mathsf{i}}(2,1)$ above cannot be done in this system since $\mathsf{h}^{\mathsf{i}}(1)$ cannot be reduced to $\langle \mathsf{h}^{\mathsf{i}}(1), [\,]\rangle$.

In order to overcome this drawback, one could *complete* the function definitions with rules that reduce each irreducible term t to a tuple of the form $\langle t, [\,]\rangle$. Although we find it a promising idea for future work, in this paper we propose a simpler approach. In the following, we consider a refinement of innermost reduction where only constructor substitutions are computed. Formally, the constructor reduction relation, $\xrightarrow{\mathsf{c}}$, is defined as follows: given ground terms $s, t \in \mathcal{T}(\mathcal{F})$, we have $s \xrightarrow{\mathsf{c}}_{\mathcal{R}} t$ iff there exist a position p in s such that no proper subterms of $s|_p$ are reducible, a rewrite rule $l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n} \in \mathcal{R}$, and a ground *constructor* substitution σ such that $s|_p = l\sigma$, $s_i\sigma \xrightarrow{\mathsf{c}}{}^*_{\mathcal{R}} t_i\sigma$ for all $i = 1, \ldots, n$, and $t = s[r\sigma]_p$. Note that the results in the previous section also hold for $\xrightarrow{\mathsf{c}}$.

In the following, given a basic term $t = \mathsf{f}(\overline{s})$, we denote by t^{i} the term $\mathsf{f}^{\mathsf{i}}(\overline{s})$. Now, we introduce our injectivization transformation as follows:

Definition 31 (injectivization). Let $\mathcal{R}$ be a pcDCTRS. We produce a new CTRS $\mathbf{I}(\mathcal{R})$ by replacing each rule $\beta : l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n}$ of $\mathcal{R}$ by a new rule of the form

$$l^{\mathsf{i}} \to \langle r, \beta(\overline{y}, \overline{w_n})\rangle \Leftarrow \overline{s_n^{\mathsf{i}} \twoheadrightarrow \langle t_n, w_n\rangle}$$

in $\mathbf{I}(\mathcal{R})$, where $\{\overline{y}\} = (\mathcal{V}\mathsf{ar}(l) \backslash \mathcal{V}\mathsf{ar}(r, \overline{s_n}, \overline{t_n})) \cup \bigcup_{i=1}^{n} \mathcal{V}\mathsf{ar}(t_i) \backslash \mathcal{V}\mathsf{ar}(r, \overline{s_{i+1,n}})$ and $\overline{w_n}$ are fresh variables. Here, we assume that the variables of $\overline{y}$ are in lexicographic order.

Observe that now we do not need to keep a trace in each term, but only a single trace term since all reductions finish in one step in a pcDCTRS. The relation between the original trace terms and the information stored in the injectivized system is formalized as follows:

Definition 32. Given a trace term $\pi = \beta(\{\overline{y_m \mapsto t_m}\}, \pi_1, \ldots, \pi_n)$, we define $\widehat{\pi}$ recursively as follows: $\widehat{\pi} = \beta(\overline{t_m}, \widehat{\pi_1}, \ldots, \widehat{\pi_n})$, where we assume that the variables $\overline{y_m}$ are in lexicographic order.

Moreover, in order to simplify the notation, we consider that a a trace term π and a singleton list of the form $[\pi]$ denote the same object. The correctness of the injectivization transformation is stated as follows:

Theorem 33. *Let $\mathcal{R}$ be a pcDCTRS and $\mathcal{R}_f = \mathbf{I}(\mathcal{R})$ be its injectivization. Then $\mathcal{R}_f$ is a pcDCTRS and, given a basic ground term s, we have $\langle s, [\,]\rangle \xrightarrow{\mathsf{c}}_{\mathcal{R}} \langle t, \pi\rangle$ iff $s^{\mathtt{i}} \xrightarrow{\mathsf{c}}_{\mathcal{R}_f} \langle t, \widehat{\pi}\rangle$.*

Proof. The fact that $\mathcal{R}_f$ is a pcDCTRS is trivial. Regarding the second part, we proceed as follows:

($\Rightarrow$) We proceed by induction on the depth k of the step $\langle s, [\,]\rangle \xrightarrow{\mathsf{c}}_{\mathcal{R}_k} \langle t, \pi\rangle$. Since the depth $k = 0$ is trivial, we consider the inductive case $k > 0$. Thus, there is a rule $\beta : l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n} \in \mathcal{R}$, and a substitution σ such that $s = l\sigma$, $\langle s_i\sigma, [\,]\rangle \xrightarrow{\mathsf{c}}_{\mathcal{R}_{k_i}} \langle t_i\sigma, \pi_i\rangle$, $i = 1, \ldots, n$, $t = r\sigma$, $\sigma' = \sigma\restriction_{(\mathcal{V}\mathsf{ar}(l)\setminus\mathcal{V}\mathsf{ar}(r,\overline{s_n},\overline{t_n}))\cup\bigcup_{i=1}^{n}\mathcal{V}\mathsf{ar}(t_i)\setminus\mathcal{V}\mathsf{ar}(r,\overline{s_{i+1,n}})}$, and $\pi = \beta(\sigma', \pi_1, \ldots, \pi_n)$. By definition of $\rightharpoonup_{\mathcal{R}_k}$, we have that $k_i < k$ for all $i = 1, \ldots, n$ and, thus, by the induction hypothesis, we have $(s_i\sigma)^{\mathtt{i}} \xrightarrow{\mathsf{c}}_{\mathcal{R}_f} \langle t_i\sigma, \widehat{\pi}_i\rangle$ for all $i = 1, \ldots, n$. Consider now the equivalent rule in $\mathcal{R}_f$: $l^{\mathtt{i}} \to \langle r, \beta(\overline{y}, \overline{w_n})\rangle \Leftarrow s_1^{\mathtt{i}} \twoheadrightarrow \langle t_1, w_1\rangle, \ldots, s_n^{\mathtt{i}} \twoheadrightarrow \langle t_n, w_n\rangle$. Therefore, we have $s^{\mathtt{i}} \xrightarrow{\mathsf{c}}_{\mathcal{R}_f} \langle t, \beta(\overline{y}\sigma, \widehat{\pi}_1, \ldots, \widehat{\pi}_n)\rangle$ where $\{\overline{y}\} = (\mathcal{V}\mathsf{ar}(l)\setminus\mathcal{V}\mathsf{ar}(r, \overline{s_n}, \overline{t_n}))\cup\bigcup_{i=1}^{n}\mathcal{V}\mathsf{ar}(t_i)\setminus\mathcal{V}\mathsf{ar}(r, \overline{s_{i+1,n}})$ and, thus, we can conclude that $\widehat{\pi} = \beta(\overline{y}\sigma, \widehat{\pi}_1, \ldots, \widehat{\pi}_n)$.

($\Leftarrow$) This direction is analogous. We proceed by induction on the depth k of the step $s^{\mathtt{i}} \xrightarrow{\mathsf{c}}_{\mathcal{R}_{f_k}} \langle t, \widehat{\pi}\rangle$. Since the depth $k = 0$ is trivial, we consider the inductive case $k > 0$. Thus, there is a rule $l^{\mathtt{i}} \to \langle r, \beta(\overline{y}, \overline{w_n})\rangle \Leftarrow s_1^{\mathtt{i}} \twoheadrightarrow \langle t_1, w_1\rangle, \ldots, s_n^{\mathtt{i}} \twoheadrightarrow \langle t_n, w_n\rangle$ in $\mathcal{R}_f$ and a substitution θ such that $l^{\mathtt{i}}\theta = s^{\mathtt{i}}$, $s_i^{\mathtt{i}}\theta \xrightarrow{\mathsf{c}}_{\mathcal{R}_{f_{k_i}}} \langle t_i, w_i\rangle\theta$, $i = 1, \ldots, n$, and $\langle r, \beta(\overline{y}, \overline{w_n})\rangle\theta = \langle t, \widehat{\pi}\rangle$. Assume that σ is the restriction of θ to the variables of the rule, excluding the fresh variables $\overline{w_n}$, and that $w_i\theta = \widehat{\pi}_i$ for all $i = 1, \ldots, n$. Therefore, $\langle s_i, [\,]\rangle\theta = \langle s_i\sigma, [\,]\rangle$ and $\langle t_i, w_i\rangle\theta = \langle t_i\sigma, \widehat{\pi}_i\rangle$, $i = 1, \ldots, n$. Then, by definition of $\mathcal{R}_{f_{k_i}}$, we have that $k_i < k$ for all $i = 1, \ldots, n$ and, thus, by the induction hypothesis, we have $\langle s_i\sigma, [\,]\rangle \xrightarrow{\mathsf{c}}_{\mathcal{R}} \langle t_i\sigma, \pi_i\rangle$, $i = 1, \ldots, n$. Consider now the equivalent rule in $\mathcal{R}$: $\beta : l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n} \in \mathcal{R}$. Therefore, we have $\langle s, [\,]\rangle \xrightarrow{\mathsf{c}}_{\mathcal{R}} \langle t, \pi\rangle$,

$\sigma' = \sigma\restriction_{(\mathcal{V}\mathsf{ar}(l)\setminus\mathcal{V}\mathsf{ar}(r,\overline{s_n},\overline{t_n}))\cup\bigcup_{i=1}^{n}\mathcal{V}\mathsf{ar}(t_i)\setminus\mathcal{V}\mathsf{ar}(r,\overline{s_{i+1,n}})}$, and $\pi = \beta(\sigma', \pi_1, \ldots, \pi_n)$. Finally, since $\{\overline{y}\} = (\mathcal{V}\mathsf{ar}(l)\setminus\mathcal{V}\mathsf{ar}(r, \overline{s_n}, \overline{t_n})) \cup \bigcup_{i=1}^{n} \mathcal{V}\mathsf{ar}(t_i)\setminus\mathcal{V}\mathsf{ar}(r, \overline{s_{i+1,n}})$, we can conclude that $\widehat{\pi} = \pi$. □

5.2. Inversion

Given an injectivized system, inversion basically amounts to switching the left- and right-hand sides of the rule and of every equation in the condition, as follows:

Definition 34 (inversion). Let $\mathcal{R}$ be a pcDCTRS and $\mathcal{R}_f = \mathbf{I}(\mathcal{R})$ be its injectivization. The inverse system $\mathcal{R}_b = \mathbf{I}^{-1}(\mathcal{R}_f)$ is obtained from $\mathcal{R}_f$ by replacing each rule[12]

$$\mathsf{f}^{\mathsf{i}}(\overline{s_0}) \to \langle r, \beta(\overline{y}, \overline{w_n})\rangle \Leftarrow \mathsf{f}_1^{\mathsf{i}}(\overline{s_1}) \twoheadrightarrow \langle t_1, w_1\rangle, \ldots, \mathsf{f}_n^{\mathsf{i}}(\overline{s_n}) \twoheadrightarrow \langle t_n, w_n\rangle$$

of $\mathcal{R}_f$ by a new rule of the form

$$\mathsf{f}^{-1}(r, \beta(\overline{y}, \overline{w_n})) \to \langle \overline{s_0}\rangle \Leftarrow \mathsf{f}_n^{-1}(t_n, w_n) \twoheadrightarrow \langle \overline{s_n}\rangle, \ldots, \mathsf{f}_1^{-1}(t_1, w_1) \twoheadrightarrow \langle \overline{s_1}\rangle$$

in $\mathbf{I}^{-1}(\mathcal{R}_f)$, where the variables of $\overline{y}$ are in lexicographic order.

Example 35. Consider again the pcDCTRS of Example 16. Here, injectivization returns the following pcDCTRS $\mathbf{I}(\mathcal{R}) = \mathcal{R}_f$:

$$\begin{aligned}
\mathsf{f}^{\mathsf{i}}(x, y, m) &\to \langle \mathsf{s}(w), \beta_1(m, x, w_1, w_2)\rangle \\
&\qquad \Leftarrow \mathsf{h}^{\mathsf{i}}(x) \twoheadrightarrow \langle x, w_1\rangle, \mathsf{g}^{\mathsf{i}}(y, \mathsf{4}) \twoheadrightarrow \langle w, w_2\rangle \\
\mathsf{h}^{\mathsf{i}}(\mathsf{0}) &\to \langle \mathsf{0}, \beta_2\rangle \\
\mathsf{h}^{\mathsf{i}}(\mathsf{1}) &\to \langle \mathsf{1}, \beta_3\rangle \\
\mathsf{g}^{\mathsf{i}}(x, y) &\to \langle x, \beta_4(y)\rangle
\end{aligned}$$

Then, inversion with $\mathbf{I}^{-1}$ produces the following pcDCTRS $\mathbf{I}^{-1}(\mathbf{I}(\mathcal{R})) = \mathcal{R}_b$:

$$\begin{aligned}
\mathsf{f}^{-1}(\mathsf{s}(w), \beta_1(m, x, w_1, w_2)) &\to \langle x, y, m\rangle \\
&\qquad \Leftarrow \mathsf{g}^{-1}(w, w_2) \twoheadrightarrow \langle y, \mathsf{4}\rangle, \mathsf{h}^{-1}(x, w_1) \twoheadrightarrow \langle x\rangle \\
\mathsf{h}^{-1}(\mathsf{0}, \beta_2) &\to \langle \mathsf{0}\rangle \\
\mathsf{h}^{-1}(\mathsf{1}, \beta_3) &\to \langle \mathsf{1}\rangle \\
\mathsf{g}^{-1}(x, \beta_4(y)) &\to \langle x, y\rangle
\end{aligned}$$

Finally, the correctness of the inversion transformation is stated as follows:

[12]Here, we assume that $\overline{s_0}$, $\overline{s_1}$,…, $\overline{s_n}$ denote arbitrary sequences of terms, i.e., $\overline{s_0} = s_{0,1}, \ldots, s_{0,l_0}$, $\overline{s_1} = s_{1,1}, \ldots, s_{1,l_1}$, etc. We use this notation for clarity.

Theorem 36. *Let $\mathcal{R}$ be a pcDCTRS, $\mathcal{R}_f = \mathbf{I}(\mathcal{R})$ its injectivization, and $\mathcal{R}_b = \mathbf{I}^{-1}(\mathcal{R}_f)$ the inversion of $\mathcal{R}_f$. Then, $\mathcal{R}_b$ is a basic pcDCTRS and, given a basic ground term $\mathsf{f}(\overline{s})$ and a constructor ground term t with $\langle t, \pi\rangle$ a safe pair, we have $\langle t, \pi\rangle \overset{\mathsf{c}}{\leftarrow}_{\mathcal{R}} \langle \mathsf{f}(\overline{s}), [\,]\rangle$ iff $\mathsf{f}^{-1}(t, \widehat{\pi}) \overset{\mathsf{c}}{\rightarrow}_{\mathcal{R}_b} \langle \overline{s}\rangle$.*

Proof. The fact that $\mathcal{R}_f$ is a pcDCTRS is trivial. Regarding the second part, we proceed as follows.

($\Rightarrow$) We proceed by induction on the depth k of the step $\langle t, \pi\rangle \overset{\mathsf{c}}{\leftarrow}_{\mathcal{R}_k} \langle \mathsf{f}(\overline{s}), [\,]\rangle$. Since the depth $k = 0$ is trivial, we consider the inductive case $k > 0$. Let $\pi = \beta(\sigma', \overline{\pi_n})$. Thus, we have that $\langle t, \beta(\sigma', \overline{\pi_n})\rangle$ is a safe pair, there is a rule $\beta : \mathsf{f}(\overline{s_0}) \to r \Leftarrow \mathsf{f}_1(\overline{s_1}) \twoheadrightarrow t_1, \ldots, \mathsf{f}_n(\overline{s_n}) \twoheadrightarrow t_n$ and substitution θ with $\mathcal{D}\mathsf{om}(\theta) = (\mathcal{V}\mathsf{ar}(r, \overline{s_1}, \ldots, \overline{s_n}) \backslash \mathcal{D}\mathsf{om}(\sigma'))$ such that $t = r\theta$, $\langle t_i\theta\sigma', \pi_i\rangle \overset{\mathsf{c}}{\rightarrow}_{\mathcal{R}_{k_i}} \langle \mathsf{f}(\overline{s_i})\theta\sigma', [\,]\rangle$ for all $i = 1, \ldots, n$, and $\mathsf{f}(\overline{s}) = \mathsf{f}(\overline{s_0})\theta\sigma'$. Note that $\overline{s_0}, \ldots, \overline{s_n}$ denote sequences of terms of arbitrary length, i.e., $\overline{s_0} = s_{0,1}, \ldots, s_{0,l_0}$, $\overline{s_1} = s_{1,1}, \ldots, s_{1,l_1}$, etc. Since $\langle t, \pi\rangle$ is a safe pair, we have that $\mathcal{D}\mathsf{om}(\sigma') = (\mathcal{V}\mathsf{ar}(\overline{s_0}) \backslash \mathcal{V}\mathsf{ar}(r, \overline{s_1}, \ldots, \overline{s_n}, \overline{t_n})) \cup \bigcup_{i=1}^{n} \mathcal{V}\mathsf{ar}(t_i) \backslash \mathcal{V}\mathsf{ar}(r, \overline{s_{i+1}}, \ldots, \overline{s_n})$. By definition of $\leftarrow_{\mathcal{R}_k}$, we have that $k_i < k$ for all $i = 1, \ldots, n$ and, by the induction hypothesis, we have $\mathsf{f}^{-1}(t_i\sigma, \widehat{\pi}_i) \overset{\mathsf{c}}{\rightarrow}_{\mathcal{R}_b} \langle \overline{s_i}\sigma\rangle$ for all $i = 1, \ldots, n$. Let us now consider the equivalent rule in $\mathcal{R}_b$:

$$\mathsf{f}^{-1}(r, \beta(\overline{y}, \overline{w_n}))) \to \langle \overline{s_0}\rangle \Leftarrow \mathsf{f}_n^{-1}(t_n, w_n) \twoheadrightarrow \langle \overline{s_n}\rangle, \ldots, \mathsf{f}_1^{-1}(t_1, w_1) \twoheadrightarrow \langle \overline{s_1}\rangle$$

Hence, we have $\mathsf{f}^{-1}(t, \beta(\overline{y}\sigma, \widehat{\pi}_1, \ldots, \widehat{\pi}_1)) \to_{\mathcal{R}_b} \langle \overline{s_0}\sigma\rangle = \langle \overline{s}\rangle$, where

$$\{\overline{y}\} = (\mathcal{V}\mathsf{ar}(\overline{s_0}) \backslash \mathcal{V}\mathsf{ar}(r, \overline{s_1}, \ldots, \overline{s_n}, \overline{t_n})) \cup \bigcup_{i=1}^{n} \mathcal{V}\mathsf{ar}(t_i) \backslash \mathcal{V}\mathsf{ar}(r, \overline{s_{i+1}}, \ldots, \overline{s_n})$$

and, thus, we can conclude that $\widehat{\pi} = \beta(\overline{y}\sigma, \widehat{\pi}_1, \ldots, \widehat{\pi}_n)$.

($\Leftarrow$) This direction is analogous. We proceed by induction on the depth k of the step $\mathsf{f}^{-1}(t, \widehat{\pi}) \overset{\mathsf{c}}{\rightarrow}_{\mathcal{R}_{b_k}} \langle \overline{s}\rangle$. Since the depth $k = 0$ is trivial, we consider the inductive case $k > 0$. Thus, there is a rule $\mathsf{f}^{-1}(r, \beta(\overline{y}, \overline{w_n}))) \to \langle \overline{s_0}\rangle \Leftarrow \mathsf{f}_n^{-1}(t_n, w_n) \twoheadrightarrow \langle \overline{s_n}\rangle, \ldots, \mathsf{f}_1^{-1}(t_1, w_1) \twoheadrightarrow \langle \overline{s_1}\rangle$ in $\mathcal{R}_b$ and a substitution θ such that $\mathsf{f}^{-1}(r, \beta(\overline{y}, \overline{w_n}))\theta = \mathsf{f}^{-1}(t, \widehat{\pi})$, $\mathsf{f}_i^{-1}(t_i, w_i)\theta \overset{\mathsf{c}}{\rightarrow}_{\mathcal{R}_{b_{k_i}}} \langle \overline{s_i}\rangle\theta$, $i = n, \ldots, 1$, and $\mathsf{f}^{-1}(r, ws)\theta = \langle \overline{s}\rangle$. Assume that σ is the restriction of θ to the variables of the rule, excluding the fresh variables $\overline{w_n}$, and that $w_i\theta = \widehat{\pi}_i$ for all $i = 1, \ldots, n$. Therefore, $\mathsf{f}^{-1}(r, \beta(\overline{y}, \overline{w_n}))\theta = \mathsf{f}^{-1}(r\sigma, \beta(\overline{y}\sigma, \widehat{\pi}_1, \ldots, \widehat{\pi}_n)$, $\mathsf{f}_i^{-1}(t_i, w_i)\theta = \mathsf{f}_i^{-1}(t_i\sigma, \widehat{\pi}_i)$ and $\langle \overline{s_i}\rangle\theta = \langle \overline{s_i}\sigma\rangle$, $i = 1, \ldots, n$. Then, by definition of $\mathcal{R}_{b_{k_i}}$, we have that $k_i < k$ for all $i = 1, \ldots, n$ and, thus, by the induction hypothesis, we have $\langle t_i\sigma, \pi_i\rangle \overset{\mathsf{c}}{\leftarrow}_{\mathcal{R}} \langle \mathsf{f}_i(\overline{s_i}\sigma), [\,]\rangle$, $i = 1, \ldots, n$. Consider now the equivalent rule in $\mathcal{R}$: $\beta : \mathsf{f}(\overline{s_0}) \to r \Leftarrow \mathsf{f}_1(\overline{s_1}) \twoheadrightarrow t_1, \ldots, \mathsf{f}_n(\overline{s_n}) \twoheadrightarrow t_n$ in $\mathcal{R}$. Therefore, we have $\langle t, \pi\rangle \overset{\mathsf{c}}{\leftarrow}_{\mathcal{R}} \langle \mathsf{f}(\overline{s}), [\,]\rangle$,

$$\sigma' = \sigma\restriction_{(\mathcal{V}\mathsf{ar}(\overline{s_0}) \backslash \mathcal{V}\mathsf{ar}(r, \overline{s_1}, \ldots, \overline{s_n}, \overline{t_n})) \cup \bigcup_{i=1}^{n} \mathcal{V}\mathsf{ar}(t_i) \backslash \mathcal{V}\mathsf{ar}(r, \overline{s_{i+1}}, \ldots, \overline{s_n})}$$

and $\pi = \beta(\sigma', \pi_1, \ldots, \pi_n)$. Finally, since $\{\overline{y}\} = (\mathcal{V}\mathsf{ar}(\overline{s_0}) \backslash \mathcal{V}\mathsf{ar}(r, \overline{s_1}, \ldots, \overline{s_n}, \overline{t_n})) \cup \bigcup_{i=1}^{n} \mathcal{V}\mathsf{ar}(t_i) \backslash \mathcal{V}\mathsf{ar}(r, \overline{s_{i+1}}, \ldots, \overline{s_n})$, we can conclude that $\widehat{\pi} = \pi$. □

5.3. Improving the transformation for injective functions

When a function is injective, one can expect the injectivization transformation to be unnecessary. This is not generally true, since some additional syntactic conditions might also be required. Furthermore, depending on the considered setting, it can be necessary to have an injective *system*, rather than an injective function. Consider, e.g., the following simple TRS:

$$\mathcal{R} = \{\ \mathsf{f}_1 \to \mathsf{f}_2, \mathsf{f}_2 \to \mathsf{0},\ \mathsf{g}_1 \to \mathsf{g}_2,\ \mathsf{g}_2 \to \mathsf{0}\ \}$$

Here, all functions are clearly injective. However, given a reduction like $\mathsf{f}_1 \to_{\mathcal{R}} \mathsf{f}_2 \to_{\mathcal{R}} \mathsf{0}$, we do not know which rule should be applied to $\mathsf{0}$ in order to go backwards until the initial term (actually, both the second and the fourth rules are applicable in the reverse direction).

Luckily, in our context, the injectivity of a function suffices since reductions in pcDCTRSs are performed in a single step. Therefore, given a reduction of the form $\mathsf{f}^{\mathsf{i}}(\overline{s_n}) \to_{\mathcal{R}} t$, a backward computation will have the form $\mathsf{f}^{-1}(t) \to_{\mathcal{R}} \langle \overline{s_n} \rangle$, so that we know that only the inverse rules of f are applicable.

Now, we present an improvement of the injectivization transformation presented in Section 5.1 which has some similarities with that in [24]. Here, we consider that the initial system is a TRS $\mathcal{R}$ since, to the best of our knowledge, there is no reachability analysis defined for DCTRSs. In the following, given a term s, we let

$$\mathsf{range}(s) = \{t \mid s\sigma \to_{\mathcal{R}}^{*} t,\ \sigma : \mathcal{V} \mapsto \mathcal{T}(\mathcal{C}),\ \text{and } t \in \mathcal{T}(\mathcal{C})\}$$

i.e., $\mathsf{range}(s)$ returns a set with the constructor normal forms of all possible ground constructor instances of s. Although computing this set is generally undecidable, there are some overapproximations based on the use of tree automata (see, e.g., [15] and the most recent approach for innermost rewriting [16]). Let us consider that $\mathsf{range}^{\alpha}(s)$ is such an approximation, with $\mathsf{range}^{\alpha}(s) \supseteq \mathsf{range}(s)$ for all terms s. Here, we are interested in determining when the right-hand sides, r_1 and r_2, of two rules do not overlap, i.e., $\mathsf{range}(r_1) \cap \mathsf{range}(r_2) = \varnothing$. For this purpose, we will check whether $\mathsf{range}^{\alpha}(r_1) \cap \mathsf{range}^{\alpha}(r_2) = \varnothing$. Since finite tree automata are closed under intersection and the emptiness of a finite tree automata is decidable, checking the emptiness of $\mathsf{range}^{\alpha}(r_1) \cap \mathsf{range}^{\alpha}(r_2)$ is decidable and can be used to *safely* identify non-overlapping right-hand sides, i.e., if $\mathsf{range}^{\alpha}(r_1) \cap \mathsf{range}^{\alpha}(r_2) = \varnothing$,

then r_1 and r_2 are definitely non-overlapping; otherwise, they may be overlapping or non-overlapping.

Now, we summarize our method to simplify some trace terms. Given a constructor TRS $\mathcal{R}$ and a rule $\beta : l \to r \in \mathcal{R}$, we check the following conditions:

1. the right-hand side r of the rule does not overlap with the right-hand side of any other rule defining the same function;
2. the rule is non-erasing, i.e., $\mathcal{V}\mathsf{ar}(l) = \mathcal{V}\mathsf{ar}(r)$;
3. the right-hand side r contains a single occurrence of a defined function symbol, say $\mathsf{f} \in \mathcal{D}$.

If these conditions hold, then the rule has the form $l \to r[\mathsf{f}(\overline{s})]_p$ with l and $\mathsf{f}(\overline{s})$ basic terms,[13] and $r[x]_p$ and $\overline{s}$ constructor terms, where x is a fresh variable. In this case, we can safely produce the following injective version:[14]

$$l^{\mathsf{i}} \to \langle r[x]_p, w\rangle \Leftarrow \mathsf{f}^{\mathsf{i}}(\overline{s}) \twoheadrightarrow \langle x, w\rangle$$

instead of

$$l^{\mathsf{i}} \to \langle r[x]_p, \beta(w)\rangle \Leftarrow \mathsf{f}^{\mathsf{i}}(\overline{s}) \twoheadrightarrow \langle x, w\rangle$$

Let us illustrate this improved transformation with a couple of examples.

Example 37. Consider the following TRS:

$$\mathcal{R} = \{\ \mathsf{f}(\mathsf{s}(x)) \to \mathsf{g}(x),\ \mathsf{f}(\mathsf{c}(x)) \to \mathsf{h}(x),\ \mathsf{g}(x) \to \mathsf{s}(x),\ \mathsf{h}(x) \to \mathsf{c}(x)\}$$

Here, it can easily be shown that $\mathsf{range}^{\alpha}(\mathsf{g}(x)) \cap \mathsf{range}^{\alpha}(\mathsf{h}(x)) = \varnothing$, the two rules defining f are non-erasing, and both contain a single occurrence of a defined function symbol in the righ-hand sides. Therefore, our improved injectivization applies and we get the following pcDCTRS $\mathcal{R}_f$:

$$\begin{array}{ll} \mathsf{f}^{\mathsf{i}}(\mathsf{s}(x)) \to \langle y, w\rangle \Leftarrow \mathsf{g}^{\mathsf{i}}(x) \twoheadrightarrow \langle y, w\rangle & \mathsf{g}^{\mathsf{i}}(x) \to \langle \mathsf{s}(x), \beta_3\rangle \\ \mathsf{f}^{\mathsf{i}}(\mathsf{c}(x)) \to \langle y, w\rangle \Leftarrow \mathsf{h}^{\mathsf{i}}(x) \twoheadrightarrow \langle y, w\rangle & \mathsf{h}^{\mathsf{i}}(x) \to \langle \mathsf{c}(x), \beta_4\rangle \end{array}$$

In contrast, the original injectivization transformation would return the following system:

$$\begin{array}{ll} \mathsf{f}^{\mathsf{i}}(\mathsf{s}(x)) \to \langle y, \beta_1(w)\rangle \Leftarrow \mathsf{g}^{\mathsf{i}}(x) \twoheadrightarrow \langle y, w\rangle & \mathsf{g}^{\mathsf{i}}(x) \to \langle \mathsf{s}(x), \beta_3\rangle \\ \mathsf{f}^{\mathsf{i}}(\mathsf{c}(x)) \to \langle y, \beta_2(w)\rangle \Leftarrow \mathsf{h}^{\mathsf{i}}(x) \twoheadrightarrow \langle y, w\rangle & \mathsf{h}^{\mathsf{i}}(x) \to \langle \mathsf{c}(x), \beta_4\rangle \end{array}$$

[13] Note that l is a basic term since we initially consider a constructor TRS and, thus, all left-hand sides are basic terms by definition.

[14] Since $l \to r$ is non-erasing, the pcDCTRS rule $l \to r[x]_p \Leftarrow \mathsf{f}(\overline{s}) \twoheadrightarrow x$ is trivially non-erasing too (according to [32], i.e., $(\mathcal{V}\mathsf{ar}(l) \backslash \mathcal{V}\mathsf{ar}(r[x]_p, \mathsf{f}(\overline{s}), x)) \cup \mathcal{V}\mathsf{ar}(x) \backslash \mathcal{V}\mathsf{ar}(r[x]_p) = \varnothing$) and, thus, no binding should be stored during the injectivization process.

Finally, the inverse system $\mathcal{R}_b$ obtained from $\mathcal{R}_f$ using the original transformation has the following form:

$$\begin{array}{ll} \mathsf{f}^{-1}(y,w) \to \langle \mathsf{s}(x) \rangle \Leftarrow \mathsf{g}^{-1}(y,w) \twoheadrightarrow \langle x \rangle & \mathsf{g}^{-1}(\mathsf{s}(x),\beta_3) \to \langle x \rangle \\ \mathsf{f}^{-1}(y,w) \to \langle \mathsf{c}(x) \rangle \Leftarrow \mathsf{h}^{-1}(y,w) \twoheadrightarrow \langle x \rangle & \mathsf{h}^{-1}(\mathsf{c}(x),\beta_4) \to \langle x \rangle \end{array}$$

For instance, given the forward reduction $\mathsf{f}^{\mathsf{i}}(\mathsf{s}(\mathsf{0})) \to_{\mathcal{R}_f} \langle \mathsf{s}(\mathsf{0}), \beta_3 \rangle$, we can build the corresponding backward reduction: $\mathsf{f}^{-1}(\mathsf{s}(\mathsf{0}),\beta_3) \to_{\mathcal{R}_b} \langle \mathsf{s}(\mathsf{0}) \rangle$.

Note, however, that the left-hand sides of f^{-1} overlap and we should reduce the conditions in order to determine which rule to apply. Therefore, in some cases, there is a trade-off between the size of the trace terms and the complexity of the reduction steps.

The example above, though, only produces a rather limited improvement since the considered functions are not recursive. Our next example shows a much significant improvement. Here, we consider the function zip (also used in [24] to illustrate the benefits of an injectivity analysis).

Example 38. Consider the following TRS $\mathcal{R}$ defining the function zip:

$$\begin{array}{rl} \mathsf{zip}([\,], ys) & \to [\,] \\ \mathsf{zip}(xs, [\,]) & \to [\,] \\ \mathsf{zip}(x : xs, y : ys) & \to \mathsf{pair}(x,y) : \mathsf{zip}(xs, ys) \end{array}$$

Here, since the third rule is non-erasing, its right-hand side contains a single occurrence of a defined function, zip, and it does not overlap with any other right-hand side, our improved injectivization applies and we get the following pcDCTRS $\mathcal{R}_f$:

$$\begin{array}{rl} \mathsf{zip}^{\mathsf{i}}([\,], ys) & \to \langle [\,], \beta_1(ys) \rangle \\ \mathsf{zip}^{\mathsf{i}}(xs, [\,]) & \to \langle [\,], \beta_2(xs) \rangle \\ \mathsf{zip}^{\mathsf{i}}(x : xs, y : ys) & \to \langle \mathsf{pair}(x,y) : zs, w \rangle \Leftarrow \mathsf{zip}^{\mathsf{i}}(xs, ys) \twoheadrightarrow \langle zs, w \rangle \end{array}$$

In contrast, the original injectivization transformation would return the following system $\mathcal{R}'_f$:

$$\begin{array}{rl} \mathsf{zip}^{\mathsf{i}}([\,], ys) & \to \langle [\,], \beta_1(ys) \rangle \\ \mathsf{zip}^{\mathsf{i}}(xs, [\,]) & \to \langle [\,], \beta_2(xs) \rangle \\ \mathsf{zip}^{\mathsf{i}}(x : xs, y : ys) & \to \langle \mathsf{pair}(x,y) : zs, \beta_3(w) \rangle \Leftarrow \mathsf{zip}^{\mathsf{i}}(xs, ys) \twoheadrightarrow \langle zs, w \rangle \end{array}$$

It might seem a small difference, but if we call $\mathsf{zip}^{\mathsf{i}}$ with two lists of n elements, the system $\mathcal{R}'_f$ would build a trace term of the form $\beta_3(\ldots\beta_3(\beta_1(\ldots))\ldots)$ with n nested constructors β_3, while $\mathcal{R}_f$ would just build the trace term $\beta_1(\ldots)$. For large values of n, this is a significant improvement in memory usage.

6. Bidirectional Program Transformation

We illustrate a practical application of our reversibilization technique in the context of *bidirectional program transformation* (see [10] for a survey). In particular, we consider the so-called *view-update* problem. Here, we have a data structure (e.g., a database) called the *source*, which is transformed to another data structure, called the *view*. Typically, we have a *view function*, view: *Source* $\to$ *View* that takes the source and returns the corresponding view, together with an *update* function, upd: *View* $\times$ *Source* $\to$ *Source* that propagates the changes in a modified view to the original source. Two basic properties that these functions should satisfy in order to be well-behaved are the following [13]:

$$\begin{aligned} \forall s \in \mathit{Source}, \forall v \in \mathit{View}\colon\ & \mathsf{view}(\mathsf{upd}(v,s)) = v \\ \forall s \in \mathit{Source}\colon\ & \mathsf{upd}(\mathsf{view}(s), s) = s \end{aligned}$$

Bidirectionalization (first proposed in the database community [5]) basically consists in, given a view function, "bidirectionalize" it in order to derive an appropriate update function. For this purpose, first, a *view complement* function is usually defined, say view^c, so that the tupled function

$$\mathsf{view} \vartriangle \mathsf{view}^c\colon \mathit{Source} \to \mathit{View} \times \mathit{Comp}$$

becomes injective. Therefore, the update function can be defined as follows:

$$\mathsf{upd}(v,s) = (\mathsf{view} \vartriangle \mathsf{view}^c)^{-1}(v, \mathsf{view}^c(s))$$

This approach has been applied to bidirectionalize view functions in a functional language in [24].

In the following, we apply our injectivization and inversion transformations in order to produce a bidirectionalization transformation that may be useful in the context of the view-update problem (with some limitations). Let us assume that we have a view function, view, that takes a source and returns the corresponding view, and which is defined by means of a pcDCTRS. Following our approach, given the original program $\mathcal{R}$, we produce an injectivized version $\mathcal{R}_f$ and the corresponding inverse $\mathcal{R}_b$. Therefore, in principle, one can use $\mathcal{R}_f \cup \mathcal{R}_b$, which will include the functions $\mathsf{view}^{\mathtt{i}}$ and view^{-1}, to define an update function as follows:

$$\mathsf{upd}(v,s) \to s' \Leftarrow \mathsf{view}^{\mathtt{i}}(s) \twoheadrightarrow \langle v', \pi \rangle, \mathsf{view}^{-1}(v,\pi) \twoheadrightarrow \langle s' \rangle$$

where s is the original source, v is the updated view, and s', the returned value, is the corresponding updated source. Note that, in our context, the function $\mathsf{view}^{\mathtt{i}}$ is somehow equivalent to $\mathsf{view} \vartriangle \mathsf{view}^c$ above.

Let us now illustrate the bidirectionalization process with an example. Consider a particular data structure, a list of *records* of the form $\mathsf{r}(t, v)$ where t is the type of the record (e.g., book, dvd, pen, etc.) and v is its price tag. The following system defines a view function that takes a type and a list of records, and returns a list with the price tags of the records of the given type:[15]

$$
\begin{array}{rcl}
\mathsf{view}(t, \mathsf{nil}) & \to & \mathsf{nil} \\
\mathsf{view}(t, \mathsf{r}(t', v) : rs) & \to & \mathsf{val}(\mathsf{r}(t', v)) : \mathsf{view}(t, rs) \Leftarrow \mathsf{eq}(t, t') \twoheadrightarrow \mathsf{true} \\
\mathsf{view}(t, \mathsf{r}(t', v) : rs) & \to & \mathsf{view}(t, rs) \Leftarrow \mathsf{eq}(t, t') \twoheadrightarrow \mathsf{false} \\
\mathsf{eq}(\mathsf{book}, \mathsf{book}) & \to & \mathsf{true} \qquad \mathsf{eq}(\mathsf{dvd}, \mathsf{dvd}) \to \mathsf{true} \\
\mathsf{eq}(\mathsf{book}, \mathsf{dvd}) & \to & \mathsf{false} \qquad \mathsf{eq}(\mathsf{dvd}, \mathsf{book}) \to \mathsf{false} \\
\mathsf{val}(\mathsf{r}(t, v)) & \to & v
\end{array}
$$

However, this system is not a pcDCTRS. Here, we use a flattening transformation to produce the following (labeled) pcDCTRS $\mathcal{R}$ which is equivalent for constructor derivations:

$$
\begin{array}{ll}
\beta_1 : & \mathsf{view}(t, \mathsf{nil}) \to \mathsf{nil} \\
\beta_2 : & \mathsf{view}(t, \mathsf{r}(t', v) : rs) \to p : r \\
& \qquad \Leftarrow \mathsf{eq}(t, t') \twoheadrightarrow \mathsf{true}, \mathsf{val}(\mathsf{r}(t', v)) \twoheadrightarrow p, \mathsf{view}(t, rs) \twoheadrightarrow r \\
\beta_3 : & \mathsf{view}(t, \mathsf{r}(t', v) : rs) \to r \Leftarrow \mathsf{eq}(t, t') \twoheadrightarrow \mathsf{false}, \mathsf{view}(t, rs) \twoheadrightarrow r \\
\beta_4 : & \mathsf{eq}(\mathsf{book}, \mathsf{book}) \to \mathsf{true} \qquad \beta_5 : \mathsf{eq}(\mathsf{dvd}, \mathsf{dvd}) \to \mathsf{true} \\
\beta_6 : & \mathsf{eq}(\mathsf{book}, \mathsf{dvd}) \to \mathsf{false} \qquad \beta_7 : \mathsf{eq}(\mathsf{dvd}, \mathsf{book}) \to \mathsf{false} \\
\beta_8 : & \mathsf{val}(\mathsf{r}(t, v)) \to v
\end{array}
$$

Now, we can apply our injectivization transformation which returns the following pcDCTRS $\mathcal{R}_f = \mathbf{I}(\mathcal{R})$:

$$
\begin{array}{l}
\mathsf{view}^{\mathtt{i}}(t, \mathsf{nil}) \to \langle \mathsf{nil}, \beta_1(t) \rangle \\
\mathsf{view}^{\mathtt{i}}(t, \mathsf{r}(t', v) : rs) \to \langle p : r, \beta_2(w_1, w_2, w_3) \rangle \\
\quad \Leftarrow \mathsf{eq}^{\mathtt{i}}(t, t') \twoheadrightarrow \langle \mathsf{true}, w_1 \rangle, \mathsf{val}^{\mathtt{i}}(\mathsf{r}(t', v)) \twoheadrightarrow \langle p, w_2 \rangle, \mathsf{view}^{\mathtt{i}}(t, rs) \twoheadrightarrow \langle r, w_3 \rangle \\
\mathsf{view}^{\mathtt{i}}(t, \mathsf{r}(t', v) : rs) \to \langle r, \beta_3(v, w_1, w_2) \rangle \\
\quad \Leftarrow \mathsf{eq}^{\mathtt{i}}(t, t') \twoheadrightarrow \langle \mathsf{false}, w_1 \rangle, \mathsf{view}^{\mathtt{i}}(t, rs) \twoheadrightarrow \langle r, w_2 \rangle \\
\mathsf{eq}^{\mathtt{i}}(\mathsf{book}, \mathsf{book}) \to \langle \mathsf{true}, \beta_4 \rangle \qquad \mathsf{eq}^{\mathtt{i}}(\mathsf{dvd}, \mathsf{dvd}) \to \langle \mathsf{true}, \beta_5 \rangle \\
\mathsf{eq}^{\mathtt{i}}(\mathsf{book}, \mathsf{dvd}) \to \langle \mathsf{false}, \beta_6 \rangle \qquad \mathsf{eq}^{\mathtt{i}}(\mathsf{dvd}, \mathsf{book}) \to \langle \mathsf{false}, \beta_7 \rangle \\
\mathsf{val}^{\mathtt{i}}(\mathsf{r}(t, v)) \to \langle v, \beta_8(t) \rangle
\end{array}
$$

[15]For simplicity, we restrict the record types to only book and dvd.

Finally, inversion returns the following pcDCTRS $\mathcal{R}_b = \mathbf{I}(\mathcal{R}_f)$:

$$\begin{aligned}
\mathsf{view}^{-1}(\mathsf{nil}, \beta_1(t)) &\to \langle t, \mathsf{nil}\rangle\\
\mathsf{view}^{-1}(p : r, \beta_2(w_1, w_2, w_3)) &\to \langle t, \mathsf{r}(t', v) : rs\rangle\\
&\Leftarrow \mathsf{eq}^{-1}(\mathsf{true}, w_1) \twoheadrightarrow \langle t, t'\rangle, \mathsf{val}^{-1}(p, w_2) \twoheadrightarrow \langle \mathsf{r}(t', v)\rangle, \mathsf{view}^{-1}(r, w_3) \twoheadrightarrow \langle t, rs\rangle\\
\mathsf{view}^{-1}(r, \beta_3(v, w_1, w_2)) &\to \langle t, \mathsf{r}(t', v) : rs\rangle\\
&\Leftarrow \mathsf{eq}^{-1}(\mathsf{false}, w_1) \twoheadrightarrow \langle t, t'\rangle, \mathsf{view}^{-1}(r, w_2) \twoheadrightarrow \langle t, rs\rangle
\end{aligned}$$

$$\begin{aligned}
\mathsf{eq}^{-1}(\mathsf{true}, \beta_4) &\to \langle \mathsf{book}, \mathsf{book}\rangle \quad \mathsf{eq}^{-1}(\mathsf{true}, \beta_5) \to \langle \mathsf{dvd}, \mathsf{dvd}\rangle\\
\mathsf{eq}^{-1}(\mathsf{false}, \beta_6) &\to \langle \mathsf{book}, \mathsf{dvd}\rangle \quad \mathsf{eq}^{-1}(\mathsf{false}, \beta_7) \to \langle \mathsf{dvd}, \mathsf{book}\rangle\\
\mathsf{val}^{-1}(v, \beta_8(t)) &\to \langle \mathsf{r}(t, v)\rangle
\end{aligned}$$

For instance, the term $\mathsf{view}(\mathsf{book}, [\mathsf{r}(\mathsf{book}, 12), \mathsf{r}(\mathsf{dvd}, 24)])$, reduces to $[12]$ in the original system $\mathcal{R}$. Given a modified view, e.g., $[15]$, we can compute the modified source using function upd above:

$$\mathsf{upd}([\mathsf{r}(\mathsf{book}, 12), \mathsf{r}(\mathsf{dvd}, 24)],\ [15])$$

Here, we have the following subcomputations:[16]

$$\begin{aligned}
&\mathsf{view}^{\mathsf{i}}(\mathsf{book}, [\mathsf{r}(\mathsf{book}, 12), \mathsf{r}(\mathsf{dvd}, 24)])\\
&\qquad \to_{\mathcal{R}_f} \langle [12], \beta_2(\beta_4, \beta_8(\mathsf{book}), \beta_3(24, \beta_6, \beta_1(\mathsf{book})))\rangle\\
&\mathsf{view}^{-1}([15], \beta_2(\beta_4, \beta_8(\mathsf{book}), \beta_3(24, \beta_6, \beta_1(\mathsf{book}))))\\
&\qquad \to_{\mathcal{R}_b} \langle \mathsf{book}, [\mathsf{r}(\mathsf{book}, 15), \mathsf{r}(\mathsf{dvd}, 24)]\rangle
\end{aligned}$$

Thus upd returns the updated source $[\mathsf{r}(\mathsf{book}, 15), \mathsf{r}(\mathsf{dvd}, 24)]$, as expected. We note that the considered example cannot be transformed using the technique in [24], the closer to our approach, since the right-hand sides of some rules contain functions which are not *treeless*.[17] Nevertheless, one could consider a transformation from pcDCTRS to functional programs with treeless functions so that the technique in [24] becomes applicable.

Our approach can solve a view-update problem as long as the view function can be encoded in a pcDCTRS. When this is the case, the results from Section 5 guarantee that function upd is well defined. Formally analyzing the class of view functions that can be represented with a pcDCTRS is an interesting topic for further research.

7. Related Work

There is no widely accepted notion of reversible computing. In this work, we have considered one of its most popular definitions, according to which a

[16] Note that, in this case, the function view requires not only the source but also the additional parameter book.

[17] A call is *treeless* if it has the form $\mathsf{f}(x_1, \ldots, x_n)$ and $x_1, \ldots, x_n$ are different variables.

computation principle is reversible if there is a method to *undo* a (forward) computation. Moreover, we expect to get back to an *exact* past state of the computation. This is often referred to as *full reversibility*.

As we have mentioned in the introduction, some of the most promising applications of reversibility include cellular automata [28], bidirectional program transformation [24], already discussed in Section 6, reversible debugging [17], where the ability to go both forward and backward when seeking the cause of an error can be very useful for the programmer, parallel discrete event simulation [34], where reversibility is used to undo the effects of speculative computations made on a wrong assumption, quantum computing [39], where all computations should be reversible, and so forth. The interested reader can find detailed surveys in the *state of the art* reports of the different working groups of COST Action IC1405 on Reversible Computation [20].

Intuitively speaking, there are two broad approaches to reversibility from a programming language perspective:

Reversible programming languages. In this case, all constructs of the programming language are reversible. One of the most popular languages within the first approach is the reversible (imperative) language Janus [23]. The language was recently rediscovered [42, 41, 43] and has since been formalized and further developed.

Irreversible programming languages and Landauer's embedding. Alternatively, one can consider an irreversible programming language, and enhance the states with some additional information (typically, the *history* of the computation so far) so that computations become reversible. This is called *Landauer's embedding*.

In this work, we consider reversibility in the context of term rewriting. To the best of our knowledge, we have presented the first approach to reversibility in term rewriting. A closest approach was introduced by Abramsky in the context of pattern matching automata [2], though his developments could easily be applied to rewrite systems as well. In Abramsky's approach, *biorthogonality* was required to ensure reversibility, which would be a very significant restriction for term rewriting systems. Basically, biorthogonality requires that, for every pair of (different) rewrite rules $l \to r$ and $l' \to r'$, l and l' do not *overlap* (roughly, they do not unify) and r and r' do not overlap too. Trivially, the functions of a biorthogonal system are injective and, thus, computations are reversible without the need of a Landauer embedding. Therefore, Abramsky's work is aimed at defining a reversible language, in contrast to our approach that is based on defining a Landauer embedding for standard term rewriting and a general class of rewrite systems.

Defining a Landauer embedding in order to make a computation mechanism reversible has been applied in different contexts and computational models, e.g., a probabilistic guarded command language [44], a low level virtual machine [35], the call-by-name lambda calculus [19, 21], cellular automata [38, 27], combinatory logic [11], a flowchart language [41], etc.

In the context of declarative languages, we find the work by Mu *et al.* [29], where a relational reversible language is presented (in the context of bidirectional programming). A similar approach was then introduced by Matsuda *et al.* [24, 25] in the context of functional programs and bidirectional transformation. The functional programs considered in [24] can be seen as linear and *right-treeless*[18] constructor TRSs. The class of functional programs is more general in [25], which would correspond to left-linear, right-treeless TRSs. The reversibilization technique of [24, 25] includes both an injectivization stage (by introducing a *view complement* function) and an inversion stage. These methods are closely related to the transformations of injectivization and inversion that we have presented in Section 5, although we developed them from a rather different starting point. Moreover, their methods for injectivization and inversion consider a more restricted class of systems than those considered in this paper. On the other hand, they apply a number of analyses to improve the result, which explains the smaller traces in their approach. All in all, we consider that our approach gives better insights to understand the need for some of the requirements of the program transformations and the class of considered programs. For instance, most of our requirements come from the need to remove programs positions from the traces, as shown in Section 4.

Finally, [37] considers the reversible language RFUN. Similarly to Janus, computations in RFUN are reversible without the need of a Landauer embedding. The paper also presents a transformation from a simple (irreversible) functional language, FUN, to RFUN, in order to highlight how irreversibilities are handled in RFUN. The transformation has some similarities with both the approach of [24] and our improved transformation in Section 5.3; on the other hand, though, [37] also applies the Bennett *trick* [6] in order to avoid some unnecessary information.

[18] There are no nested defined symbols in the right-hand sides, and, moreover, any term rooted by a defined function in the right-hand sides can only take different variables as its proper subterms.

8. Discussion and Future Work

In this paper, we have introduced a reversible extension of term rewriting. In order to keep our approach as general as possible, we have initially considered DCTRSs as input systems, and proved the soundness and reversibility of our extension of rewriting. Then, in order to introduce a reversibilization transformation for these systems, we have also presented a transformation from DCTRSs to pure constructor systems (pcDCTRSs) which is correct for constructor reduction. A further improvement is presented for injective functions, which may have a significant impact in memory usage in some cases. Finally, we have successfully applied our approach in the context of bidirectional program transformation.

We have developed a prototype implementation of the reversibilization transformations introduced in Section 5. The tool can read an input TRS file (format `.trs` [1]) and then it applies in a sequential way the following transformations: flattening, simplification of constructor conditions, injectivization, and inversion. The tool prints out the CTRSs obtained at each transformation step. It is publicly available through a web interface from `http://kaz.dsic.upv.es/rev-rewriting.html`, where we have included a number of examples to easily test the tool.

As for future work, we plan to investigate new methods to further reduce the size of the traces. In particular, we find it interesting to define a reachability analysis for DCTRSs. A reachability analysis for CTRSs without extra-variables (1-CTRSs) can be found in [12], but the extension to deal with extra-variables in DCTRSs (since a DCTRS is a particular case of 3-CTRS) seems challenging. Furthermore, as mentioned in the paper, a completion procedure to add *default* cases to some functions (as suggested in Section 5.1) may help to broaden the applicability of the technique and avoid the restriction to constructor reduction. Finally, our injectivization and inversion transformations are correct w.r.t. innermost reduction. Extending our results to a lazy strategy is also an interesting topic for further research.

Acknowledgements

We thank the anonymous reviewers for their useful comments and suggestions to improve this paper.

References

[1] Annual international termination competition. Available from URL: `http://www.termination-portal.org/wiki/Termination_Competition`.

[2] S. Abramsky. A structural approach to reversible computation. *Theoretical Computer Science*, 347(3):441–464, 2005.

[3] J. M. Almendros-Jiménez and G. Vidal. Automatic partial inversion of inductively sequential functions. In Z. Horváth, V. Zsók, and A. Butterfield, editors, *Implementation and Application of Functional Languages, 18th International Symposium (IFL 2006), Revised Selected Papers*, volume 4449 of *Lecture Notes in Computer Science*, pages 253–270. Springer, 2007.

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

[5] F. Bancilhon and N. Spyratos. Update semantics of relational views. *ACM Transactions on Database Systems*, 6(4):557–575, 1981.

[6] C. H. Bennett. Logical reversibility of computation. *IBM Journal of Research and Development*, 17:525–532, 1973.

[7] C. H. Bennett. Notes on the history of reversible computation. *IBM Journal of Research and Development*, 44(1):270–278, 2000.

[8] J. Bergstra and J. Klop. Conditional Rewrite Rules: confluence and termination. *Journal of Computer and System Sciences*, 32:323–362, 1986.

[9] P. Crescenzi and C. H. Papadimitriou. Reversible simulation of space-bounded computations. *Theoretical Computer Science*, 143(1):159–165, 1995.

[10] K. Czarnecki, J. N. Foster, Z. Hu, R. Lämmel, A. Schürr, and J. F. Terwilliger. Bidirectional transformations: A cross-discipline perspective. In R. F. Paige, editor, *Proc. of the 2nd Int'l Conf. on Theory and Practice of Model Transformations (ICMT 2009)*, volume 5563 of *Lecture Notes in Computer Science*, pages 260–283. Springer, 2009.

[11] A. Di Pierro, C. Hankin, and H. Wiklicky. Reversible combinatory logic. *Mathematical Structures in Computer Science*, 16(4):621–637, 2006.

[12] G. Feuillade and T. Genet. Reachability in Conditional Term Rewriting Systems. *Electronic Notes in Theoretical Computer Science*, 86(1):133–146, 2003.

[13] J. N. Foster, M. B. Greenwald, J. T. Moore, B. C. Pierce, and A. Schmitt. Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem. *ACM Transactions on Programming Languages and Systems*, 29(3):17, 2007.

[14] M. P. Frank. Introduction to reversible computing: motivation, progress, and challenges. In N. Bagherzadeh, M. Valero, and A. Ramírez, editors, *Proceedings of the Second Conference on Computing Frontiers*, pages 385–390. ACM, 2005.

[15] T. Genet. Decidable Approximations of Sets of Descendants and Sets of Normal Forms. In T. Nipkow, editor, *Proc. of the 9th International Conference on Rewriting Techniques and Applications (RTA'98)*, volume 1379 of *Lecture Notes in Computer Science*, pages 151–165. Springer, 1998.

[16] T. Genet and Y. Salmon. Reachability Analysis of Innermost Rewriting. In M. Fernández, editor, *Proc. of the 26th International Conference on Rewriting Techniques and Applications (RTA'15)*, volume 36 of *LIPIcs*, pages 177–193. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

[17] E. Giachino, I. Lanese, and C. A. Mezzina. Causal-consistent reversible debugging. In S. Gnesi and A. Rensink, editors, *Proc. of the 17th International Conference on Fundamental Approaches to Software Engineering (FASE 2014)*, volume 8411 of *Lecture Notes in Computer Science*, pages 370–384. Springer, 2014.

[18] N. Hirokawa and G. Moser. Automated Complexity Analysis Based on the Dependency Pair Method. In A. Armando, P. Baumgartner, and G. Dowek, editors, *Proc. of IJCAR 2008*, volume 5195 of *Lecture Notes in Computer Science*, pages 364–379. Springer, 2008.

[19] L. Huelsbergen. A logically reversible evaluator for the call-by-name lambda calculus. In T. Toffoli and M. Biafore, editors, *Proc. of PhysComp96*, pages 159–167. New England Complex Systems Institute, 1996.

[20] COST Action IC1405 on Reversible Computation - extending horizons of computing. URL: `http://revcomp.eu/`.

[21] W. E. Kluge. A reversible SE(M)CD machine. In P. W. M. Koopman and C. Clack, editors, *Proc. of the 11th International Workshop on the Implementation of Functional Languages, IFL'99. Selected Papers*, volume 1868 of *Lecture Notes in Computer Science*, pages 95–113. Springer, 2000.

[22] R. Landauer. Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development*, 5:183–191, 1961.

[23] C. Lutz and H. Derby. Janus: A time-reversible language, 1986. A letter to R. Landauer. Available from URL `http://tetsuo.jp/ref/janus.pdf`.

[24] K. Matsuda, Z. Hu, K. Nakano, M. Hamana, and M. Takeichi. Bidirectionalization transformation based on automatic derivation of view complement functions. In R. Hinze and N. Ramsey, editors, *Proc. of the 12th ACM SIGPLAN International Conference on Functional Programming, ICFP 2007*, pages 47–58. ACM, 2007.

[25] K. Matsuda, Z. Hu, K. Nakano, M. Hamana, and M. Takeichi. Bidirectionalizing programs with duplication through complementary function derivation. *Computer Software*, 26(2):56–75, 2009. In Japanese.

[26] A. Middeldorp and E. Hamoen. Completeness results for basic narrowing. *Applicable Algebra in Engineering, Communication and Computing*, 5:213–253, 1994.

[27] K. Morita. Reversible simulation of one-dimensional irreversible cellular automata. *Theoretical Computer Science*, 148(1):157–163, 1995.

[28] K. Morita. Computation in reversible cellular automata. *International Journal of General Systems*, 41(6):569–581, 2012.

[29] S. Mu, Z. Hu, and M. Takeichi. An injective language for reversible computation. In D. Kozen and C. Shankland, editors, *Proc. of the 7th International Conference on Mathematics of Program Construction (MPC 2004)*, volume 3125 of *Lecture Notes in Computer Science*, pages 289–313. Springer, 2004.

[30] M. Nagashima, M. Sakai, and T. Sakabe. Determinization of conditional term rewriting systems. *Theoretical Computer Science*, 464:72–89, 2012.

[31] N. Nishida, A. Palacios, and G. Vidal. Reversible term rewriting. In D. Kesner and B. Pientka, editors, *Proc. of the 1st International Conference on Formal Structures for Computation and Deduction (FSCD'16)*, volume 52 of *LIPIcs*, pages 28:1–28:18. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

[32] N. Nishida, M. Sakai, and T. Sakabe. Soundness of unravelings for conditional term rewriting systems via ultra-properties related to linearity. *Logical Methods in Computer Science*, 8(3-4):1–49, Aug. 2012.

[33] N. Nishida and G. Vidal. Program inversion for tail recursive functions. In M. Schmidt-Schauß, editor, *Proceedings of the 22nd International Conference on Rewriting Techniques and Applications (RTA 2011)*, volume 10 of *LIPIcs*, pages 283–298. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.

[34] M. Schordan, D. R. Jefferson, P. D. B. Jr., T. Oppelstrup, and D. J. Quinlan. Reverse code generation for parallel discrete event simulation. In J. Krivine and J. Stefani, editors, *Proc. of the 7th International Conference on Reversible Computation (RC 2015)*, volume 9138 of *Lecture Notes in Computer Science*, pages 95–110. Springer, 2015.

[35] B. Stoddart, R. Lynas, and F. Zeyda. A virtual machine for supporting reversible probabilistic guarded command languages. *Electronic Notes in Theoretical Computer Science*, 253(6):33–56, 2010.

[36] Terese. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003.

[37] M. K. Thomsen and H. B. Axelsen. Interpretation and programming of the reversible functional language RFUN. In R. Lämmel, editor, *Proc. of the 27th Symposium on the Implementation and Application of Functional Programming Languages (IFL'15)*, pages 8:1–8:13. ACM, 2015.

[38] T. Toffoli. Computation and construction universality of reversible cellular automata. *Journal of Computer and System Sciences*, 15(2):213–231, 1977.

[39] T. Yamakami. One-way reversible and quantum finite automata with advice. *Information and Computation*, 239:122–148, 2014.

[40] T. Yokoyama. Reversible computation and reversible programming languages. *Electronic Notes in Theoretical Computer Science*, 253(6):71–81, 2010.

[41] T. Yokoyama, H. Axelsen, and R. Glück. Fundamentals of reversible flowchart languages. *Theoretical Computer Science*, 611:87–115, 2016.

[42] T. Yokoyama, H. B. Axelsen, and R. Glück. Reversible flowchart languages and the structured reversible program theorem. In *Proc. of the 35th International Colloquium on Automata, Languages and Programming (ICALP 2008)*, volume 5126 of *Lecture Notes in Computer Science*, pages 258–270. Springer, 2008.

[43] T. Yokoyama and R. Glück. A reversible programming language and its invertible self-interpreter. In G. Ramalingam and E. Visser, editors, *Proc. of the 2007 ACM SIGPLAN Workshop on Partial Evaluation and Semantics-based Program Manipulation (PEPM 2007)*, pages 144–153. ACM, 2007.

[44] P. Zuliani. Logical reversibility. *IBM Journal of Research and Development*, 45(6):807–818, 2001.