Reversible Term Rewriting*

Naoki Nishida[1], Adrián Palacios[2], and Germán Vidal[2]

1 **Graduate School of Information Science, Nagoya University**
Furo-cho, Chikusa-ku, 4648603 Nagoya, Japan
nishida@is.nagoya-u.ac.jp

2 **MiST, DSIC, Universitat Politècnica de València**
Camino de Vera, s/n, 46022 Valencia, Spain
{apalacios, gvidal}@dsic.upv.es

Abstract

Essentially, in a reversible programming language, for each forward computation step from state S to state S', there exists a constructive and deterministic 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 paper, 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 and deterministic method to get t_1 from t_2. Here, we introduce a conservative extension of term rewriting that becomes reversible. Furthermore, we also define a transformation to make a rewrite system reversible using standard term rewriting.

1998 ACM Subject Classification F.4.2 Grammars and Other Rewriting Systems

Keywords and phrases term rewriting, reversible computation, program transformation

Digital Object Identifier 10.4230/LIPIcs.FSCD.2016.<article-no>

1 Introduction

The notion of reversible computation can be traced back to Landauer's pioneering work [14]. Although Landauer was mainly concerned with the energy consumption of erasing data in irreversible computing (only recently experimentally measured [5]), 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 [3] 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 [6] 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.

* This work has been partially supported by the EU (FEDER) and the Spanish *Ministerio de Economía y Competitividad* (MINECO) under grant TIN2013-44742-C4-1-R, by the *Generalitat Valenciana* under grant PROMETEO-II/2015/013 (SmartLogic) and by the COST Action IC1405 on Reversible Computation. A. Palacios was partially supported by the the EU (FEDER) and the Spanish *Ayudas para contratos predoctorales para la formación de doctores de la Sec. Estado de Investigación, Desarrollo e Innovación del MINECO* under FPI grant BES-2014-069749. Part of this research was done while the second and third authors were visiting Nagoya University; they gratefully acknowledge their hospitality.

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 ranging from cellular automata [19] and bidirectional program transformation [15] to quantum computing [29], to name a few. We refer the interested reader to, e.g., [4, 9, 30] for a high level account of the principles of reversible computation.

In this work, we focus on *term rewriting* [2, 26], a computation model that underlies most rule-based programming languages. Essentially, there are two approaches to designing a reversible language: one can either restrict the language to only contain reversible constructs, or one can include some additional information (typically, the *history* of the computation so far) so that all constructs become reversible, which is called a *Landauer's embedding*. The first approach is considered, e.g., by Abramsky in the context of pattern matching automata [1]. There, *biorthogonality* is required to ensure reversibility, which would be a very significant restriction for term rewriting systems. Thus, we follow the second, more general approach by introducing the information required for the reductions to become reversible.

To be more precise, we introduce a general and intuitive notion of *reversible* term rewriting by following essentially a Landauer's 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 given reduction. In other words, given a rewriting reduction $s \to^*_{\mathcal{R}} t$, a 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 8). In contrast, our backward relation $\leftharpoondown_{\mathcal{R}}$ is deterministic (thus confluent) and terminating.

We then introduce a *flattening* transformation for rewrite systems 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 $\mathcal{R} = \{\mathsf{a}(\mathsf{0}, y) \to y,\ \mathsf{a}(\mathsf{s}(x), y) \to \mathsf{s}(\mathsf{a}(x, y))\}$ defining the addition on natural numbers built from constructors $\mathsf{0}$ and $\mathsf{s}(\ )$, we produce the following *basic* (conditional) system: $\mathcal{R}' = \{\mathsf{a}(\mathsf{0}, y) \to y,\ \mathsf{a}(\mathsf{s}(x), y) \to \mathsf{s}(z) \Leftarrow \mathsf{a}(x, y) \twoheadrightarrow z\}$ (see Example 16 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

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

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

where a^i is an injective version of function a, a^{-1} is the inverse of a^i, and β_1, β_2 are fresh symbols introduced to label the rules of the original system.

We consider *conditional* rewrite systems in this work, 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. We refer the interested reader to [21] for a definition of reversible term rewriting for unconditional systems.

The paper is organized as follows. After introducing some preliminaries in Section 2, we present our approach to reversible term rewriting in Section 3. Then, Section 4 introduces a transformation to *basic* systems, and Section 5 presents injectivization and inversion transformations in order to make a rewrite system reversible with standard rewriting. Finally, Section 6 discusses some related work and Section 7 concludes and points out some ideas for future research. More details and missing proofs can be found in [21].

2 Preliminaries

We assume familiarity with basic concepts of term rewriting. We refer the reader to, e.g., [2] and [26] for further details.

Terms and Substitutions. A *signature* $\mathcal{F}$ is a set of 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 $p \in \mathcal{P}os(t)$, 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. *TRSs and Rewriting.* 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 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 t is called *irreducible* or in *normal form* w.r.t. a TRS $\mathcal{R}$ if there is no term s with $t \to_\mathcal{R} s$. A substitution is called *normalized* w.r.t. $\mathcal{R}$ if every variable in the domain is replaced by a normal form w.r.t. $\mathcal{R}$. We sometimes omit "w.r.t. $\mathcal{R}$" 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.

In this paper, we 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 $l \to r \Leftarrow C$ [17]. 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 $o_1 \twoheadrightarrow o'_1, \ldots, o_n \twoheadrightarrow o'_n$ by $\overline{o_n \twoheadrightarrow o'_n}$. Moreover, we assume that rewrite rules are labelled, i.e., given a CTRS $\mathcal{R}$, we denote by $\beta : l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n}$ a rewrite rule with label β. Labels are unique in a CTRS. Also, to relate label β to fixed variables, we consider that the variables of the rewrite rules are not renamed and that the reduced terms are always ground.[1] We often write $s \to_{p,\beta} t$ instead of $s \to_{p,l \to r \Leftarrow \overline{s_n \to t_n}} t$ if rule $l \to r \Leftarrow \overline{s_n \to t_n}$ is labeled with β.

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 $i = 1, \ldots, n$. 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 t and a rule $\beta : l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n}$ with $t|_p = l\theta$, 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 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 $t \to_{p,\beta} t[r\sigma]_p$ 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:

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

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

[1] Equivalently, one could require terms to be variable disjoint with the variables of the rewrite system, but we require groundness for simplicity.

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. Theorem 10). 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. 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 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.

Reversible rewriting is then defined on pairs $\langle t, \pi \rangle$, where t is a term and π is a trace:

▶ **Definition 2** (trace)**.** Given a CTRS $\mathcal{R}$, a *trace* in $\mathcal{R}$ is recursively defined as follows:[2]

- the empty list is a trace;
- if $\pi, \pi_1, \ldots, \pi_n$ are traces in $\mathcal{R}$, $n \geq 0$, there is a rule $\beta : l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n} \in \mathcal{R}$, 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. Our trace terms have some similarities with *proof terms* [26]. However, proof terms do not store the bindings of erased variables (and, to the best of our knowledge, are only defined for unconditional TRSs).

Our reversible term rewriting relation is only defined on *safe* pairs. This notion will be clarified below, after introducing the definition of reversible rewriting.

▶ **Definition 3** (safe pair)**.** Let $\mathcal{R}$ be a DCTRS. The pair $\langle s, \pi \rangle$ is *safe* in $\mathcal{R}$ iff, for all trace terms $\beta(p, \sigma, \overline{\pi_n})$ in π, σ is a ground substitution with $\mathcal{D}\mathsf{om}(\sigma) = (\mathcal{V}\mathsf{ar}(l) \backslash \mathcal{V}\mathsf{ar}(r, \overline{s_n}, \overline{t_n})) \cup \bigcup_{i=1}^{n} \mathcal{V}\mathsf{ar}(t_i) \backslash \mathcal{V}\mathsf{ar}(r, \overline{s_{i+1,n}})$ and $\beta : l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n} \in \mathcal{R}$.

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

▶ **Definition 4** (reversible rewriting)**.** Let $\mathcal{R}$ be a DCTRS. The reversible rewrite relation $\rightharpoonup_{\mathcal{R}}$ is defined on 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:

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

iff there exist a position $p \in \mathcal{P}os(s)$, 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 $\sigma' = \sigma|_{(\mathcal{V}\mathsf{ar}(l) \backslash \mathcal{V}\mathsf{ar}(r, \overline{s_n}, \overline{t_n})) \cup \bigcup_{i=1}^{n} \mathcal{V}\mathsf{ar}(t_i) \backslash \mathcal{V}\mathsf{ar}(r, \overline{s_{i+1,n}})}$. 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 $\mathcal{D}\mathsf{om}(\theta) = \mathcal{V}\mathsf{ar}(r, \overline{s_n}) \backslash \mathcal{D}\mathsf{om}(\sigma')$, $t|_p = r\theta$, $\langle t_i\,\theta \cup \sigma', \pi_i \rangle \leftharpoondown_{\mathcal{R}}^*$

[2] As it is common, $[\,]$ denotes the empty list and $x : xs$ is a list with head x and tail xs.

$\langle s_i\,\theta{\cup}\sigma', [\,]\rangle$ for all $i = 1, \ldots, n$, and $s = t[l\,\theta{\cup}\sigma']_p$. Here, we assume that $\cup$ is the union of substitutions and that it binds stronger than substitution application, i.e., $l\,\theta{\cup}\sigma' = l(\theta \cup \sigma')$. Note that $\theta \cup \sigma'$ is well defined since $\mathcal{D}\mathsf{om}(\theta) \cap \mathcal{D}\mathsf{om}(\sigma') = \varnothing$ (actually, $\theta \cup \sigma' = \theta\sigma' = \sigma'\theta$ since they are also ground).

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

▶ **Example 5.** Consider the DCTRS $\mathcal{R}$ from Example 1. Given the term $\mathsf{double}(\mathsf{s}(\mathsf{s}(0)))$, we have, for instance, the following forward derivation:

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

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

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

The reader can easily construct the associated backward derivation:

$$\langle \mathsf{add}(\mathsf{s}(\mathsf{s}(0)), \mathsf{s}(\mathsf{s}(0))), [\beta_1(1.1, id), \beta_2(1, id), \beta_2(\epsilon, id), \beta_3(\epsilon, id, \pi)]\rangle \leftharpoondown^*_{\mathcal{R}} \langle \mathsf{double}(\mathsf{s}(\mathsf{s}(0))), [\,]\rangle$$

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 rewrite rule:

- 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 11 for more details.

▶ **Example 6.** Consider, e.g., the following DCTRS:

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

and the step $\langle \mathsf{f}(0, 2, 4), [\,]\rangle \rightharpoonup_{\mathcal{R}} \langle \mathsf{s}(2), [\beta_1(\epsilon, \sigma', \pi_1, \pi_2)]\rangle$ with $\sigma' = \{m \mapsto 4, x \mapsto 0\}$, $\pi_1 = [\beta_2(\epsilon, id)]$ and $\pi_2 = [\beta_4(\epsilon, \{y \mapsto 4\})]$. 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 $\langle \mathsf{s}(2), [\beta_1(\epsilon, \sigma', \pi_1, \pi_2)]\rangle$. If the binding for x were unknown, this step would not be deterministic. As mentioned above, an instantiated reverse rule $(\mathsf{s}(w) \to \mathsf{f}(x, y, m) \Leftarrow w \twoheadrightarrow \mathsf{g}(y, 4), x \twoheadrightarrow \mathsf{h}(x))\sigma' = \mathsf{s}(w) \to \mathsf{f}(0, y, 4) \Leftarrow w \twoheadrightarrow \mathsf{g}(y, 4), 0 \twoheadrightarrow \mathsf{h}(0)$ would be a DCTRS 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 should first compute the dependencies between the equations in the conditions), so we prefer to keep the simpler conditions for DCTRSs, which is still quite a general class of CTRSs.

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 7.** *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 $\langle s, \pi\rangle \rightharpoonup_{\mathcal{R}}^* \langle t, \pi'\rangle$.*

We note that this is not the case for the backward relation: $t \leftarrow_{\mathcal{R}} s$ does not imply $\langle t, \pi'\rangle \leftharpoonup_{\mathcal{R}} \langle s, \pi\rangle$ for an arbitrary trace π'.[3] This is actually the purpose of our notion of reversible rewriting: $\leftharpoonup_{\mathcal{R}}$ 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. Nonetheless, one can still ensure that, for any step $t \leftarrow_{\mathcal{R}} s$, there is a trace π' such that $\langle t, \pi'\rangle \leftharpoonup_{\mathcal{R}} \langle s, \pi\rangle$ for some trace π (which is an easy consequence of Theorems 7 and 10).

▶ **Example 8.** Consider the following simple 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).

The following result states that every pair which is reachable from an initial pair with an empty trace is safe, and follows easily by induction on the length of the derivations:

▶ **Proposition 9.** *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 $\mathcal{R}$.*

For the following result, we need some preliminary notions (see, e.g., [26]). For every oriented CTRS $\mathcal{R}$, we inductively define the TRSs $\mathcal{R}_k$, $k \geq 0$, as follows:

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

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 $\rightharpoonup_{\mathcal{R}}$, $\leftharpoonup_{\mathcal{R}}$, and $\rightleftharpoons_{\mathcal{R}}$.

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

▶ **Theorem 10.** *Let $\mathcal{R}$ be a DCTRS. Given the safe pairs $\langle s, \pi\rangle$ and $\langle t, \pi'\rangle$, for all $k, m \geq 0$, $\langle s, \pi\rangle \rightharpoonup_{\mathcal{R}_k}^m \langle t, \pi'\rangle$ iff $\langle t, \pi'\rangle \leftharpoonup_{\mathcal{R}_k}^m \langle s, \pi\rangle$.*

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 $\langle s, \pi\rangle \rightharpoonup_{\mathcal{R}_k}^m \langle t, \pi'\rangle$. Since the base case is trivial, we consider the inductive case $(k, m) > (0, 0)$. Consider a derivation $\langle s, \pi\rangle \rightharpoonup_{\mathcal{R}_k}^{m-1} \langle s_0, \pi_0\rangle \rightharpoonup_{\mathcal{R}_k} \langle t, \pi'\rangle$ whose associated product is (k, m). By Proposition 9, 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 $\langle s_0, \pi_0\rangle \leftharpoonup_{\mathcal{R}_k}^{m-1} \langle s, \pi\rangle$. Consider now the step $\langle s_0, \pi_0\rangle \rightharpoonup_{\mathcal{R}_k} \langle t, \pi'\rangle$. Thus, there exist a

[3] Here, and in the following, we assume that $\leftarrow_{\mathcal{R}} = (\to_{\mathcal{R}})^{-1}$.

position $p \in \mathcal{P}os(s_0)$, 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 $\sigma = \theta \cup \sigma'$ is well defined. 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 $s'_0 = t[l\,\theta\cup\sigma']_p = t[l\sigma]_p = s_0$, 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 $(k, m) > (0, 0)$. Let us 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 9, 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 $s_0|_p = r\theta$, $\langle t_i\,\theta\cup\sigma', \pi_i\rangle \leftharpoondown^*_{\mathcal{R}_{k'_i}} \langle s_i\,\theta\cup\sigma', [\,]\rangle$ for all $i = 1, \ldots, n$, and $s = s_0[l\,\theta\cup\sigma']_p$. Moreover, since $\langle s_0, \pi_0\rangle$ is safe, we have that $\mathcal{D}\mathsf{om}(\sigma') = (\mathcal{V}\mathsf{ar}(l)\setminus\mathcal{V}\mathsf{ar}(r, \overline{s_n}, \overline{t_n})) \cup \bigcup_{i=1}^{n} \mathcal{V}\mathsf{ar}(t_i)\setminus\mathcal{V}\mathsf{ar}(r, \overline{s_{i+1,n}})$ and, thus, $\theta \cup \sigma'$ is well defined. 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. Hence, by the induction hypothesis, we have $\langle s_i\,\theta\cup\sigma', [\,]\rangle \rightharpoonup^*_{\mathcal{R}_{k'_i}} \langle t_i\,\theta\cup\sigma', \pi_i\rangle$ for all $i = 1, \ldots, n$. Let $\sigma = \theta \cup \sigma'$, which is well defined since $\mathcal{D}\mathsf{om}(\theta) \cap \mathcal{D}\mathsf{om}(\sigma') = \varnothing$. 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 $s'_0 = s[r\sigma]_p = s[r\,\theta\cup\sigma']_p$; moreover, $s[r\,\theta\cup\sigma']_p = s[r\theta]_p = s_0[r\theta]_p = s_0$ since $\mathcal{D}\mathsf{om}(\sigma') \cap \mathcal{V}\mathsf{ar}(r) = \varnothing$, which concludes the proof. ◀

The relevance of our backward relation $\leftharpoondown_{\mathcal{R}}$ stems from the fact that it is deterministic (thus confluent), terminating, and has a constructive definition. 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.

▶ **Theorem 11.** *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 that $m = 0$ is trivial, and thus we let $m > 0$. Assume $\langle t, \pi'\rangle \leftharpoondown^{m-1}_{\mathcal{R}} \langle u, \pi''\rangle$. If there is no step using $\leftharpoondown_{\mathcal{R}}$ from $\langle u, \pi''\rangle$, the claim follows trivially for all $m' \leq m$. Now, assume there is at least one step issuing from $\langle u, \pi''\rangle$, e.g., $\langle u, \pi''\rangle \leftharpoondown_{\mathcal{R}} \langle s, \pi\rangle$. For the base case $k = 1$, the applied rule is unconditional and we prove that this is the only possible step. By definition, we have $\pi'' = \beta(p, \sigma') : \pi$, $p \in \mathcal{P}os(u)$, $\beta : l \to r \in \mathcal{R}_1$, and there exists a ground substitution θ with $\mathcal{D}\mathsf{om}(\theta) = \mathcal{V}\mathsf{ar}(r)$ such that $u|_p = r\theta$ and $s = u[l\,\theta\cup\sigma']_p$. The only source of nondeterminism may come from choosing a rule labeled with β and from the computation of the substitution θ. The claim trivially follows since labels are unique in $\mathcal{R}$ and, if there is another ground substitution θ' with $\theta' = \mathcal{V}\mathsf{ar}(r)$ and $u|_p = r\theta'$, then $\theta = \theta'$.

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 $u|_p = r\theta$, $\langle t_i\,\theta\cup\sigma', \pi_i\rangle \leftharpoondown^*_{\mathcal{R}_j} \langle s_i\,\theta\cup\sigma', [\,]\rangle$, $j < k$,

for all $i = 1, \ldots, n$, and $s = t[l\,\theta{\cup}\sigma']_p$. By the induction hypothesis, the subderivations $\langle t_i\,\theta{\cup}\sigma', \pi_i\rangle \leftharpoondown^*_{\mathcal{R}_j} \langle s_i\,\theta{\cup}\sigma', [\,]\rangle$ are deterministic, i.e., $\langle s_i\,\theta{\cup}\sigma', [\,]\rangle$ is a unique resulting term obtained by reducing $\langle t_i\,\theta{\cup}\sigma', \pi_i\rangle$. 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 the rule $r\sigma' \to l\sigma' \Leftarrow t_n\sigma' \twoheadrightarrow s_n\sigma', \ldots, t_1\sigma' \twoheadrightarrow s_1\sigma'$ 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. Regarding the first refinement, 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 work, 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 (cf. Section 5).[4]

In the following, rather than restricting the class of considered systems, 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* [11] 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. Now, we introduce the following subclass of DCTRSs:

▶ **Definition 12** (basic DCTRS). A DCTRS $\mathcal{R}$ is called *basic* if, for any rule $l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n} \in \mathcal{R}$, we have that r, $\overline{s_n}$ and $\overline{t_n}$ are either basic or constructor terms.

In principle, any DCTRS can be transformed into a basic DCTRS by applying a sequence of *flattening* transformations. Roughly speaking, flattening involves transforming a term 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.

[4] 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) program instrumentation.

▶ **Definition 13** (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 $p \in \mathcal{P}os(s_i)$, $1 \leqslant i \leqslant n$, or $l \to r[w]_q \Leftarrow \overline{s_n \twoheadrightarrow t_n}, r|_q \twoheadrightarrow w$, for some $q \in \mathcal{P}os(r)$, where w is a fresh variable.[5] Then, a CTRS $\mathcal{R}'$ is obtained from $\mathcal{R}$ by a *flattening* step if $\mathcal{R}' = (\mathcal{R} \backslash \{R\}) \cup \{R'\}$.

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 w.r.t. *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 $t = s[r\sigma]_p$.

▶ **Theorem 14.** *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, we have $s \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} t$ iff $s \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}'} t$.*

Therefore, both a DCTRS and its basic version —obtained by applying a sequence of flattening steps— are equivalent w.r.t. innermost reduction. This justifies our use of basic DCTRSs in the remainder of this paper.

A nice property of basic DCTRSs is that one can consider reductions only at *topmost* positions. Formally, given a DCTRS $\mathcal{R}$, 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 $\xrightarrow{\epsilon}$ for standard rewriting, and $\xrightarrow{\epsilon}_{\mathcal{R}}, \xleftarrow{\epsilon}_{\mathcal{R}}$ for our reversible rewrite relations.

The following result basically states that $\xrightarrow{\mathsf{i}}$ and $\xrightarrow{\epsilon}$ are equivalent for basic DCTRSs:

▶ **Theorem 15.** *Let $\mathcal{R}$ be a DCTRS and $\mathcal{R}'$ be a basic DCTRS obtained from $\mathcal{R}$ by a sequence of flattening steps. Given ground terms s, t such that s is basic, we have $s \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} t$ iff $s \xrightarrow{\epsilon}{}^*_{\mathcal{R}'} t$.*

Therefore, when considering basic DCTRSs 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 16.** Consider the following TRS $\mathcal{R}$ defining addition and multiplication on natural numbers, and its associated basic DCTRS $\mathcal{R}'$:

$$\begin{array}{lll}
\mathcal{R} = \{ & \beta_1 : \ \mathsf{a}(0, y) \to y, & \mathcal{R}' = \{ \ \beta'_1 : \ \mathsf{a}(0, y) \to y, \\
& \beta_2 : \ \mathsf{a}(\mathsf{s}(x), y) \to \mathsf{s}(\mathsf{a}(x, y)), & \beta'_2 : \ \mathsf{a}(\mathsf{s}(x), y) \to \mathsf{s}(z) \Leftarrow \mathsf{a}(x, y) \twoheadrightarrow z, \\
& \beta_3 : \ \mathsf{m}(0, y) \to 0, & \beta'_3 : \ \mathsf{m}(0, y) \to 0, \\
& \beta_4 : \mathsf{m}(\mathsf{s}(x), y) \to \mathsf{a}(\mathsf{m}(x, y), y) \ \} & \beta'_4 : \mathsf{m}(\mathsf{s}(x), y) \to \mathsf{a}(z, y) \Leftarrow \mathsf{m}(x, y) \twoheadrightarrow z \ \}
\end{array}$$

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

$$\mathsf{m}(\mathsf{s}(0), \mathsf{s}(0)) \xrightarrow{\mathsf{i}}_{\mathcal{R}} \mathsf{a}(\mathsf{m}(0, \mathsf{s}(0)), \mathsf{s}(0)) \xrightarrow{\mathsf{i}}_{\mathcal{R}} \mathsf{a}(0, \mathsf{s}(0)) \xrightarrow{\mathsf{i}}_{\mathcal{R}} \mathsf{s}(0)$$

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

$$\mathsf{m}(\mathsf{s}(0), \mathsf{s}(0)) \xrightarrow{\epsilon}_{\mathcal{R}'} \mathsf{a}(0, \mathsf{s}(0)) \xrightarrow{\epsilon}_{\mathcal{R}'} \mathsf{s}(0) \quad \text{with } \mathsf{m}(0, \mathsf{s}(0)) \xrightarrow{\epsilon}_{\mathcal{R}'} 0$$

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

Trivially, all results in Section 3 hold for basic DCTRSs and top reductions starting from basic terms. The simpler trace terms without positions 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 basic 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 $\leftharpoondown_{\mathcal{R}}$. Therefore, $\mathcal{R}_f$ can be seen as an *injectivization* of $\mathcal{R}$, and $\mathcal{R}_b$ can be seen as the *inversion* of $\mathcal{R}_f$.

5.1 Injectivization

Essentially, injectivization in our context amounts to add the traces to the rewrite rules, so that standard rewriting can be used:

▶ **Definition 17** (injectivization)**.** Let $\mathcal{R}$ be a basic DCTRS. We produce a new CTRS $\mathcal{I}(\mathcal{R})$ by replacing each rule $\beta : l \to r \Leftarrow s_1 \twoheadrightarrow t_1, \ldots, s_n \twoheadrightarrow t_n$ of $\mathcal{R}$ by a new rule of the form

$$\langle l, ws\rangle \to \langle r, \beta(\overline{y}, \overline{w_n}) : ws\rangle \Leftarrow \langle s_1, [\,]\rangle \twoheadrightarrow \langle t_1, w_1\rangle, \ldots, \langle s_n, [\,]\rangle \twoheadrightarrow \langle t_n, w_n\rangle$$

in $\mathcal{I}(\mathcal{R})$, 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 both ws and $\overline{w_n}$ are fresh variables. Here, we assume that the variables of $\overline{y}$ are in lexicographic order.

Observe that there is a clear correspondence with the notion of reversible rewriting by only assuming that the reduced positions are always ϵ and, thus, they are not stored in the trace. Note also that, rather than storing a substitution, as in $\beta(\sigma, \pi_1, \ldots, \pi_n)$, we add the variables of interest to the trace term, $\beta(\overline{y}, \pi_1, \ldots, \pi_n)$, where $\overline{y}$ represent the domain of σ.

▶ **Example 18.** Consider again the DCTRS $\mathcal{R}$ from Example 6, which is already a basic DCTRS. Then, $\mathcal{R}_f = \mathcal{I}(\mathcal{R})$ is as follows:[6]

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

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

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

As can be seen in the example above, the trace terms that occur in a reversible rewrite derivation with a basic DCTRS $\mathcal{R}$ and those that occur in a top reduction with $\mathcal{I}(\mathcal{R})$ are similar but not exactly equal. We formalize their relation as follows:

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

▶ **Definition 19.** Given a trace π, we define $\widehat{\pi}$ recursively as follows:

$$\widehat{\pi} \;=\; \begin{cases} [\,] & \text{if } \pi = [\,] \\ \beta(\overline{t_m}, \widehat{\pi_1}, \ldots, \widehat{\pi_n}) & \text{if } \pi = \beta(\{y_1 \mapsto t_1, \ldots, y_m \mapsto t_m\}, \pi_1, \ldots, \pi_n) \end{cases}$$

where we assume that the variables $\overline{y_m}$ are in lexicographic order.

The following result states the correctness of our injectivization transformation:

▶ **Theorem 20.** *Let $\mathcal{R}$ be a basic DCTRS and $\mathcal{R}_f = \mathcal{I}(\mathcal{R})$ be its injectivization. Then $\mathcal{R}_f$ is a basic DCTRS and, given a basic ground term s, we have $\langle s, [\,]\rangle \xrightarrow{\epsilon}{}^*_{\mathcal{R}} \langle t, \pi\rangle$ iff $\langle s, [\,]\rangle \xrightarrow{\epsilon}{}^*_{\mathcal{R}_f} \langle t, \widehat{\pi}\rangle$.*

5.2 Inversion

In general, function inversion is a difficult and often undecidable problem (see, e.g., [24, 22, 10, 23]). For injectivized systems, though, it becomes straightforward:

▶ **Definition 21** (inversion)**.** Let $\mathcal{R}$ be a basic DCTRS and let $\mathcal{R}_f = \mathcal{I}(\mathcal{R})$ be its injectivization. Then, the inverse system, $\mathcal{R}_b = \mathcal{I}^{-1}(\mathcal{R}_f)$ is obtained from $\mathcal{R}_f$ by transforming every rule

$$\langle l, ws\rangle \to \langle r, \beta(\overline{y}, \overline{w_n}) : ws\rangle \Leftarrow \langle s_1, [\,]\rangle \twoheadrightarrow \langle t_1, w_1\rangle, \ldots, \langle s_n, [\,]\rangle \twoheadrightarrow \langle t_n, w_n\rangle$$

into a rule of the form

$$\langle r, \beta(\overline{y}, \overline{w_n}) : ws\rangle^{-1} \to \langle l, ws\rangle^{-1} \Leftarrow \langle t_n, w_n\rangle^{-1} \twoheadrightarrow \langle s_n, [\,]\rangle^{-1}, \ldots, \langle t_1, w_1\rangle^{-1} \twoheadrightarrow \langle s_1, [\,]\rangle^{-1}$$

We use a different symbol $\langle \_, \_\rangle^{-1}$ since we may want to use both the forward and the backward functions in the same system.

▶ **Example 22.** Consider the DCTRS $\mathcal{R}_f$ from Example 18. Then, its inversion $\mathcal{R}_b = \mathcal{I}^{-1}(\mathcal{R}_f)$ is defined as follows:

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

The correctness of the inversion transformation is then stated as follows:

▶ **Theorem 23.** *Let $\mathcal{R}$ be a basic DCTRS, $\mathcal{R}_f = \mathcal{I}(\mathcal{R})$ its injectivization, and $\mathcal{R}_b = \mathcal{I}^{-1}(\mathcal{R}_f)$ the inversion of $\mathcal{R}_f$. Then, $\mathcal{R}_b$ is a basic DCTRS and, given a basic or constructor ground term t and a trace π with $\langle t, \pi\rangle$ safe, we have $\langle t, \pi\rangle \xleftarrow{\epsilon}{}^*_{\mathcal{R}} \langle s, [\,]\rangle$ iff $\langle t, \widehat{\pi}\rangle^{-1} \xrightarrow{\epsilon}{}^*_{\mathcal{R}_b} \langle s, [\,]\rangle^{-1}$.*

5.3 An Improved Reversibilization Procedure

Using the transformations introduced so far, given a DCTRS $\mathcal{R}$, we can produce a basic DCTRS $\mathcal{R}'$, which can then be injectivized $\mathcal{I}(\mathcal{R}')$ and reversed $\mathcal{I}^{-1}(\mathcal{I}(\mathcal{R}'))$. Although one can find several applications for $\mathcal{I}(\mathcal{R}')$ and $\mathcal{I}^{-1}(\mathcal{I}(\mathcal{R}'))$, we note that these systems are aimed at mimicking the reversible relations $\rightharpoonup_{\mathcal{R}'}$ and $\leftharpoondown_{\mathcal{R}'}$, rather than computing injective and inverse versions of the *functions* defined in $\mathcal{R}'$. In other words, $\mathcal{I}(\mathcal{R}')$ defines a single function $\langle \_, \_\rangle$ and $\mathcal{I}^{-1}(\mathcal{I}(\mathcal{R}'))$ a single function $\langle \_, \_\rangle^{-1}$. Now, we refine these transformations so that one can actually produce injective and inverse versions of the original functions.

In principle, one could consider that the injectivization of a rule of the form[7]

$$\beta : \mathsf{f}(\overline{s_0}) \to r \Leftarrow \mathsf{f}_1(\overline{s_1}) \twoheadrightarrow t_1, \ldots, \mathsf{f}_n(\overline{s_n}) \twoheadrightarrow t_n$$

will produce the following rule

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

where traces are now added as an additional argument of each function. The following example, though, illustrates that this is not correct in general.

▶ **Example 24.** Consider the following basic DCTRS $\mathcal{R}$:

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

together with the following top reduction:

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

The improved injectivization above would return the following basic DCTRS:

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

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

In order to solve the above 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 $t = s[r\sigma]_p$.

Furthermore, we also require a further requirement on DCTRSs: we say that $\mathcal{R}$ is a c-DCTRS (a pure-constructor system [20]) if $\mathcal{R}$ is a DCTRS and, for any rule $l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n}$, we have that $l, \overline{s_n}$ are basic terms and $r, \overline{t_n}$ are constructor terms. Note that requiring $\overline{s_n}$ to be basic terms (thus excluding constructor terms) is not a real restriction since any equation of the form $s \twoheadrightarrow t$, with s (and t) a constructor term, can be removed by matching s and t, removing the equation, and applying the matching substitution to the rule (cf. [23]).

▶ **Definition 25** (refined injectivization)**.** Let $\mathcal{R}$ be a basic c-DCTRS. We produce a new CTRS $\mathbf{I}(\mathcal{R})$ by replacing each rule $\beta : \mathsf{f}(\overline{s_0}) \to r \Leftarrow \mathsf{f}_1(\overline{s_1}) \twoheadrightarrow t_1, \ldots, \mathsf{f}_n(\overline{s_n}) \twoheadrightarrow t_n$ of $\mathcal{R}$ by a new rule of the form

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

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.

[7] By abuse, here we let $\overline{s_0}, \ldots, \overline{s_n}$ denote sequences of terms of arbitrary length.

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 basic c-DCTRS. By abuse of notation, we still use the notation $\widehat{\pi}$ when π is a trace term instead of a trace.

▶ **Theorem 26.** *Let $\mathcal{R}$ be a basic c-DCTRS and $\mathcal{R}_f = \mathbf{I}(\mathcal{R})$ be its injectivization. Then $\mathcal{R}_f$ is a basic c-DCTRS and, given a basic ground term $\mathsf{f}(\overline{s})$ and a constructor ground term t, we have $\langle \mathsf{f}(\overline{s}), [\,] \rangle \xrightarrow{\mathsf{c}}_{\mathcal{R}} \langle t, \pi \rangle$ iff $\mathsf{f}^i(\overline{s}) \xrightarrow{\mathsf{c}}_{\mathcal{R}_f} \langle t, \widehat{\pi} \rangle$.*

Now, the refined version of the inversion transformation proceeds as follows:

▶ **Definition 27** (refined inversion)**.** Let $\mathcal{R}$ be a basic c-DCTRS 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

$$\mathsf{f}^i(\overline{s_0}) \to \langle r, \beta(\overline{y}, \overline{w_n}) \rangle \Leftarrow \mathsf{f}^i_1(\overline{s_1}) \twoheadrightarrow \langle t_1, w_1 \rangle, \ldots, \mathsf{f}^i_n(\overline{s_n}) \twoheadrightarrow \langle t_n, w_n \rangle$$

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

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

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

▶ **Example 28.** Consider again the basic DCTRS of Example 6 which is a c-DCTRS. The injectivization transformation $\mathbf{I}$, returns the following c-DCTRS $\mathcal{R}_f$:

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

Then, inversion with $\mathbf{I}^{-1}$ produces the following c-DCTRS $\mathcal{R}_b$:

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

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

▶ **Theorem 29.** *Let $\mathcal{R}$ be a basic c-DCTRS, $\mathcal{R}_f = \mathbf{I}(\mathcal{R})$ 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 c-DCTRS 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 $\langle t, \pi \rangle \xleftarrow{\mathsf{c}}_{\mathcal{R}} \langle \mathsf{f}(\overline{s}), [\,] \rangle$ iff $\mathsf{f}^{-1}(t, \widehat{\pi}) \xrightarrow{\mathsf{c}}_{\mathcal{R}_b} \langle \overline{s} \rangle$.*

5.4 Applications

A potential application of our reversibilization technique is in the context of *bidirectional program transformation* (see, e.g., [8, 15] and references therein). This technique applies when we have a data structure, called the *source*, which is transformed to another data structure, called the *view*. Typically, we have a *view function* that takes the source and returns the corresponding view. Here, the bidirectionalization transformation aims at defining a *backward transformation* that takes a modified view, and returns the corresponding *modified* source. Defining a view function and a backward transformation that form a bidirectional transformation is not easy, and therefore our reversibilization technique can be useful in this context. For instance, let us assume that we have a view function, view, that takes a source and returns the corresponding view, and is defined by means of a c-DCTRS. Then, following

our approach, we can produce an injective version, say view^i, and an inverse version view^{-1}. Now, one could solve the view update problem with the following function:

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

where s is the original source, v' is the updated view, and s' —the returned value— is the corresponding updated source.

A more challenging application is the *reversibilization of cellular automata* [19]. In a cellular automaton, evolution is determined by some fixed rule (generally, a mathematical function) that determines the new state of each cell in terms of the current state of the cell and the states of the cells in its neighborhood. If we consider a rewrite system for defining this rule, our approach can help us to produce *reversible* cellular automata from irreversible ones, an important feature in this field.

As in [18], we can consider a one-dimensional irreversible cellular automaton where each cell has two neighbours. The cellular automaton can be represented as a (potentially infinite) array of cells. Consider, e.g., the following function to control the evolution of a cellular automaton whose cells can only take value □ or ■ (it is known as *rule 150*):

$$\begin{array}{llll} \beta_1 : \mathsf{f}(\square,\square,\square) \to \square & \beta_2 : \mathsf{f}(\square,\square,\blacksquare) \to \blacksquare & \beta_3 : \mathsf{f}(\square,\blacksquare,\square) \to \blacksquare & \beta_4 : \mathsf{f}(\square,\blacksquare,\blacksquare) \to \square \\ \beta_5 : \mathsf{f}(\blacksquare,\square,\square) \to \blacksquare & \beta_6 : \mathsf{f}(\blacksquare,\square,\blacksquare) \to \square & \beta_7 : \mathsf{f}(\blacksquare,\blacksquare,\square) \to \square & \beta_8 : \mathsf{f}(\blacksquare,\blacksquare,\blacksquare) \to \blacksquare \end{array}$$

Evolution takes place then by applying *simultaneously* the above function to every cell and its neighbours. Following our approach, we can injectivize function f as follows:

$$\begin{array}{llll} \mathsf{f}^i(\square,\square,\square) \to \langle \square, \beta_1 \rangle & \mathsf{f}^i(\square,\square,\blacksquare) \to \langle \blacksquare, \beta_2 \rangle & \mathsf{f}^i(\square,\blacksquare,\square) \to \langle \blacksquare, \beta_3 \rangle & \mathsf{f}^i(\square,\blacksquare,\blacksquare) \to \langle \square, \beta_4 \rangle \\ \mathsf{f}^i(\blacksquare,\square,\square) \to \langle \blacksquare, \beta_5 \rangle & \mathsf{f}^i(\blacksquare,\square,\blacksquare) \to \langle \square, \beta_6 \rangle & \mathsf{f}^i(\blacksquare,\blacksquare,\square) \to \langle \square, \beta_7 \rangle & \mathsf{f}^i(\blacksquare,\blacksquare,\blacksquare) \to \langle \blacksquare, \beta_8 \rangle \end{array}$$

We can then consider that cells include a list of rule labels, so that the following evolution:

$$[\ \square,\ \square,\ \blacksquare,\ \square,\ \square\ ] \Rightarrow [\ \square,\ \blacksquare,\ \blacksquare,\ \blacksquare,\ \square\ ] \Rightarrow [\ \blacksquare,\ \square,\ \blacksquare,\ \square,\ \blacksquare\ ] \Rightarrow \cdots$$

with the original cellular automaton, are now represented as follows:

$$\begin{array}{ll} & [\ \langle \square, [\,] \rangle, \langle \square, [\,] \rangle, \langle \blacksquare, [\,] \rangle, \langle \square, [\,] \rangle, \langle \square, [\,] \rangle\ ] \\ \Rightarrow & [\ \langle \square, [\beta_1] \rangle, \langle \blacksquare, [\beta_2] \rangle, \langle \blacksquare, [\beta_3] \rangle, \langle \blacksquare, [\beta_5] \rangle, \langle \square, [\beta_1] \rangle\ ] \\ \Rightarrow & [\ \langle \blacksquare, [\beta_2, \beta_1] \rangle, \langle \square, [\beta_4, \beta_2] \rangle, \langle \blacksquare, [\beta_8, \beta_3] \rangle, \langle \square, [\beta_7, \beta_5] \rangle, \langle \blacksquare, [\beta_5, \beta_1] \rangle\ ] \\ \Rightarrow & \cdots \end{array}$$

Thus, the new cellular automaton is clearly reversible. With respect to the previous approach in [18], our reversibilization process is more intuitive (the one proposed in [18] is rather *ad-hoc*), does not increase the number of steps (while in [18] $2n + k$ steps are required for each step of the original cellular automaton, where k is a constant and n is the number of non-empty cells in the cellular automaton), and its correctness is trivial by construction (correctness is only sketched in [18]). On the other hand, our approach increases the size of the cellular automaton by a factor that depends in principle on the length of the computation (but not on the size of the cellular automaton).

A more detailed description of the above applications can be found in [21].

6 Related Work

Regarding reversible computing, one can already find a number of references in the literature (e.g., [4, 9, 30]). Our work starts with the well-known approach of Landauer [14] which

proposes that saving the history of a computation makes it reversible. This approach to reversibilization has already been considered in the past and has been applied in different contexts and computational models, e.g., a probabilistic guarded command language [32], a low level virtual machine [25], the call-by-name lambda calculus [12, 13], cellular automata [28, 18], combinatory logic [7], a flowchart language [31], or a functional language [15, 27].

However, to the best of our knowledge, this is the first work that considers a reversible extension of (conditional) term rewriting. We note, though, that Abramsky [1] introduced an approach to reversible computation with *pattern matching automata*, which could also be represented in terms of standard notions of term rewriting. His approach, though, requires a condition called *biorthogonality* (which, in particular, implies injectivity), a condition that would be overly restrictive in our setting. Roughly speaking, in our approach we achieve a similar class of systems through injectivization from more general systems.

Another related work are the papers by Matsuda et al [15, 16] which focus on *bidirectional program transformation* for functional programs. In [15], functional programs corresponding to linear and *right-treeless*[8] constructor TRSs are considered. In [16], the previous class is extended to those corresponding to left-linear right-treeless TRSs. The methods in [15, 16] 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. Besides being more general, we consider that our approach gives better insights to understand the need for the requirements of the program transformations. Finally, [27] introduces a transformation for functional programs which has some similarities with both the approach of [15] and our improved transformation in Section 5.3; in contrast, though, [27] also applies the Bennett *trick* [3] in order to avoid some unnecessary information.

7 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 basic DCTRSs which is correct for innermost reduction. Finally, for constructor reduction, we are able to further refine our reversibilization transformations. We have successfully applied our approach in the context of bidirectional program transformation and the reversibilization of cellular automata.

As for future work, we plan to investigate restricted classes of CTRSs so that we can further reduce the size of the traces. In particular, we will look for conditions under which we can remove the variable bindings, the rule label, or even the complete trace. For this purpose, we will consider non-erasing rules and injective functions, since we think that there are different contexts where these conditions arise quite naturally.

Acknowledgements

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

[8] 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.

References

1 S. Abramsky. A structural approach to reversible computation. *Theor. Comput. Sci.*, 347(3):441–464, 2005.

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

3 C. H. Bennet. Logical reversibility of computation. *IBM Journal of Research and Development*, 17:525–532, 1973.

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

5 A. Bérut, A. Arakelyan, A. Petrosyan, S. Ciliberto, R. Dillenschneider, and E. Lutz. Experimental verification of Landauer's principle linking information and thermodynamics. *Nature*, 483:187–189, 2012.

6 P. Crescenzi and C. H. Papadimitriou. Reversible simulation of space-bounded computations. *Theor. Comput. Sci.*, 143(1):159–165, 1995.

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

8 N. Foster, K. Matsuda, and J. Voigtländer. Three complementary approaches to bidirectional programming. In Jeremy Gibbons, editor, *Generic and Indexed Programming - International Spring School, SSGIP 2010, Oxford, UK, March 22-26, 2010, Revised Lectures*, volume 7470 of *Lecture Notes in Computer Science*, pages 1–46. Springer, 2012.

9 M. P. Frank. Introduction to reversible computing: motivation, progress, and challenges. In Nader Bagherzadeh, Mateo Valero, and Alex Ramírez, editors, *Proceedings of the Second Conference on Computing Frontiers*, pages 385–390. ACM, 2005.

10 R. Glück and M. Kawabe. A method for automatic program inversion based on LR(0) parsing. *Fundam. Inform.*, 66(4):367–395, 2005.

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

12 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.

13 W. E. Kluge. A reversible SE(M)CD machine. In Pieter W. M. Koopman and Chris 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.

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

15 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.

16 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.

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

18 K. Morita. Reversible simulation of one-dimensional irreversible cellular automata. *Theor. Comput. Sci.*, 148(1):157–163, 1995.

19 K. Morita. Computation in reversible cellular automata. *Int. J. General Systems*, 41(6):569–581, 2012.

20 M. Nagashima, M. Sakai, and T. Sakabe. Determinization of conditional term rewriting systems. *Theor. Comput. Sci.*, 464:72–89, 2012.

21 N. Nishida, A. Palacios, and G. Vidal. Reversible term rewriting: foundations and applications. Technical report, DSIC, UPV, 2016. Available from the following URL: `http://users.dsic.upv.es/~gvidal/german/rr16/`.

22 N. Nishida, M. Sakai, and T. Sakabe. Partial inversion of constructor term rewriting systems. In Jürgen Giesl, editor, *Proceedings of the 16th International Conference on Rewriting Techniques and Applications (RTA 2005)*, volume 3467 of *Lecture Notes in Computer Science*, pages 264–278. Springer, 2005.

23 N. Nishida and G. Vidal. Program inversion for tail recursive functions. In Manfred 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.

24 A. Romanenko. The generation of inverse functions in Refal. In *Proc. of IFIP TC2 Workshop on Partial Evaluation and Mixed Computation*, pp. 427–444, North-Holland, 1988.

25 B. Stoddart, R. Lynas, and F. Zeyda. A virtual machine for supporting reversible probabilistic guarded command languages. *Electr. Notes Theor. Comput. Sci.*, 253(6):33–56, 2010.

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

27 M. K. Thomsen and H. B. Axelsen. Interpretation and programming of the reversible functional language RFUN. In *Proc. of the 27th International Symposium on Implementation and Application of Functional Languages (IFL 2015)*. Springer, 2016. To appear.

28 T. Toffoli. Computation and construction universality of reversible cellular automata. *J. Comput. Syst. Sci.*, 15(2):213–231, 1977.

29 T. Yamakami. One-way reversible and quantum finite automata with advice. *Inf. Comput.*, 239:122–148, 2014.

30 T. Yokoyama. Reversible computation and reversible programming languages. *Electr. Notes Theor. Comput. Sci.*, 253(6):71–81, 2010.

31 T. Yokoyama, H.B. Axelsen, and R. Glück. Fundamentals of reversible flowchart languages. *Theor. Comput. Sci.*, 611:87–115, 2016.

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