# Inverse Unfold Problem and Its Heuristic Solving

**Masanori Nagashima, Tomofumi Kato, Masahiko Sakai, and Naoki Nishida**

**Graduate School of Information Science, Nagoya University**
**Furo-cho, Chikusa-ku, Nagoya 464–8603 Japan**
nagashima@sakabe.i.is.nagoya-u.ac.jp, tomofumi@trs.cm.is.nagoya-u.ac.jp,
sakai@is.nagoya-u.ac.jp, nishida@is.nagoya-u.ac.jp

## Abstract

Unfold/fold transformations have been widely studied in various programming paradigms and are used in program transformations, theorem proving, and so on. This paper, by using an example, show that restoring an one-step unfolding is not easy, i.e., a challenging task, since some rules used by unfolding may be lost. We formalize this problem by regarding one-step program transformation as a relation. Next we discuss some issues on a specific framework, called pure-constructor systems, which constitute a subclass of conditional term rewriting systems. We show that the inverse of $T$ preserves rewrite relations if $T$ preserves rewrite relations and the signature. We propose a heuristic procedure to solve the problem, and show its successful examples. We improve the procedure, and show examples for which the improvement takes effect.

**1998 ACM Subject Classification** I.2.2 Automatic Programming

**Keywords and phrases** program transformation, unfolding, conditional term rewriting system

**Digital Object Identifier** 10.4230/OASIcs.WPTE.2014.27

## 1 Introduction

Unfold/fold transformations have been widely studied on functional[6, 23], logic[11, 24, 25, 21, 22, 20] and constraint logic [12, 7, 4, 8] programs. They are used in program transformations, theorem proving, and so on.

This paper proposes the inverse problem of one-step unfolding. Let's see that the problem is not trivial by an example in terms of term rewriting systems (TRSs). Both TRSs $\mathcal{R}_1$ and $\mathcal{R}_2$, given as follows, define the same function $\mathsf{mult}$ that computes the multiplication of two natural numbers:

$$\mathcal{R}_1 = \left\{ \begin{array}{l} \mathsf{mult}(0, y) \to 0, \\ \mathsf{mult}(\mathsf{s}(x), y) \to \mathsf{add}(\mathsf{mult}(x, y), y) \end{array} \right\} \cup \mathcal{R}_{\mathsf{add}}, \text{ and}$$

$$\mathcal{R}_2 = \left\{ \begin{array}{l} \mathsf{mult}(0, y) \to 0, \\ \mathsf{mult}(\mathsf{s}(0), y) \to \mathsf{add}(0, y), \\ \mathsf{mult}(\mathsf{s}^2(x), y) \to \mathsf{add}(\mathsf{add}(\mathsf{mult}(x, y), y), y) \end{array} \right\} \cup \mathcal{R}_{\mathsf{add}},$$

where

$$\mathcal{R}_{\mathsf{add}} = \left\{ \begin{array}{l} \mathsf{add}(0, y) \to y, \\ \mathsf{add}(\mathsf{s}(x), y) \to \mathsf{s}(\mathsf{add}(x, y)) \end{array} \right\}.$$

Here $\mathcal{R}_2$ is derived from $\mathcal{R}_1$ by unfolding $\mathsf{mult}(x, y)$ in the right-hand side of the second rewrite rule in $\mathcal{R}_1$ by using the rules for $\mathsf{mult}$ in $\mathcal{R}_1$. On the other hand, however, it is difficult to transform $\mathcal{R}_2$ into $\mathcal{R}_1$ in the reverse direction. One may think a folding operation is applicable for this purpose, but it is impossible because the second rewrite rule of $\mathcal{R}_1$ is necessary for the folding, but is missing in $\mathcal{R}_2$.

This paper is organized as follows. First, we define the *inverse problem* of an one-step program transformation in Section 3. In sessions that follow Section 3, we discuss some issues on a specific framework, called pure-constructor system, used in our previous work [14] on a determinization of conditional term rewriting systems. We targeted this framework because of the firmness of the structure of the rules, i.e., every root position of left-hand sides in the body or conditions of rewrite rules is a defined symbol, and all the other position have no defined symbols even in right-hand sides. Remark that a deterministic pure-constructor system is convertible to an equivalent TRS, vice versa. Nested defined symbols in a right-hand side of a rule of a TRS are represented by a sequence of conditions.

In Section 4, we show that the inverse of an one-step transformation $T$ preserves rewrite relations if $T$ preserves rewrite relations and their signatures. In Section 5, we propose a heuristic procedure for this problem and show some examples. Overcoming failure examples, we propose an advanced heuristic solving in Section 6 and demonstrate its effectiveness for those examples. Finally, in Section 7, we show a motivated example induced from the program inversion[10, 15, 16, 17, 18].

## 2 Preliminaries

In this section, we introduce notations used in this paper. We assume that the reader is familiar with basic concepts of term rewriting [2, 19].

Let $\mathcal{F}$ be a *signature*, a finite set of *function symbols* accompanied with a mapping *arity* which maps each function symbol $f$ to a natural number $\mathrm{arity}(f)$. $\mathcal{F}$ is assumed to be partitioned into two disjoint sets $\mathcal{D}$ of *defined symbols* and $\mathcal{C}$ of *constructors*, that is, $\mathcal{F} = \mathcal{D} \uplus \mathcal{C}$. Let $\mathcal{V}$ be a countably infinite set of *variables* such that $\mathcal{F} \cap \mathcal{V} = \emptyset$. A function symbol $g \in \mathcal{F}$ is called a *constant* if $\mathrm{arity}(g) = 0$.

The set of *terms* over $\mathcal{F}$ and $\mathcal{V}$ is denoted by $\mathcal{T}(\mathcal{F}, \mathcal{V})$, the set of variables occurring at least one of terms $t_1, \ldots, t_n$ by $\mathcal{V}ar(t_1, \ldots, t_n)$. A term $t \in \mathcal{T}(\mathcal{F}, \emptyset)$ is called *ground*. The set of all ground terms is denoted by $\mathcal{T}(\mathcal{F})$. A term $t \in \mathcal{T}(\mathcal{C}, \mathcal{V})$ is called a *constructor term*. A term of the form $f(t_1, \ldots, t_n)$ is called a *pattern* in $\mathcal{T}(\mathcal{F}, \mathcal{V})$ if $f \in \mathcal{D}$ and $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{C}, \mathcal{V})$. The *root symbol* of a term $t$ is denoted by $\mathrm{root}(t)$.

Let $\Box \notin \mathcal{F} \cup \mathcal{V}$ be a special constant, called a *hole*. A *context* is a term $C \in \mathcal{T}(\mathcal{F} \cup \{\Box\}, \mathcal{V})$ with exactly one occurrence of $\Box$. We write $C[t]$ for the term obtained from $C$ by replacing the occurrence of $\Box$ in $C$ with a term $t$.

A *substitution* is a function $\sigma : \mathcal{V} \to \mathcal{T}(\mathcal{F}, \mathcal{V})$ such that $\{x \mid \sigma(x) \neq x\}$ is finite. The set $\{x \mid \sigma(x) \neq x\}$ is denoted by $\mathcal{D}om(\sigma)$ and called the *domain* of $\sigma$. A substitution $\sigma$ can be extended to $\sigma : \mathcal{T}(\mathcal{F}, \mathcal{V}) \to \mathcal{T}(\mathcal{F}, \mathcal{V})$ in a natural way. We write $t\sigma$ for $\sigma(t)$. A substitution $\sigma$ is *ground* if $x\sigma \in \mathcal{T}(\mathcal{F})$ for all $x \in \mathcal{D}om(\sigma)$. A substitution $\sigma$ is a *constructor substitution* if $x\sigma \in \mathcal{T}(\mathcal{C}, \mathcal{V})$ for all $x \in \mathcal{D}om(\sigma)$. The *composition* $\sigma\theta$ of two substitutions $\sigma$ and $\theta$ is defined as $x(\sigma\theta) = (x\sigma)\theta$.

An *equation* is a pair of terms $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, which is denoted by $s \sim t$. A substitution $\sigma$ is a *unifier* of a set $E$ of equations if $s\sigma = t\sigma$ for every $s \sim t \in E$. $E$ is said to be *unifiable* if there is a unifier of $E$. A unifier $\sigma$ of $E$ is a *most general unifier* of $E$ if for any unifier $\theta$ of $E$, there exists a substitution $\delta$ such that $\theta = \sigma\delta$. It is known that most general unifiers of $E$ are unique up to variable renaming. We use $\mathrm{mgu}(E)$ for the most general unifier of $E$.

An *oriented conditional rewrite rule* (*rewrite rule*, for short) is a formula in the form of $l \to r \Leftarrow u_1 \to v_1; \cdots; u_n \to v_n$ $(n \geq 0)$. $l \to r$ and $u_1 \to v_1; \cdots; u_n \to v_n$ are called the *body part* and the *conditional part* of the rule, respectively. Terms $l$ and $r$ are called the *left-hand side* and the *right-hand side* of the rule, respectively. Each $u_i \to v_i$ $(1 \le i \le n)$ is called a *condition* of the rule. A rewrite rule whose conditional part is empty is called an *unconditional rule*, denoted as $l \to r$ by omitting $\Leftarrow$. The set of variables occurring in an object $o$ (e.g., a rewrite rule) is denoted by $\mathcal{V}ar(o)$.

A condition $u \to v$ is called a *pattern condition* if $u$ is a pattern and $v$ is a constructor term. A rewrite rule $l \to r \Leftarrow c$ is called a *pure-constructor rule* if $l$ is a pattern, $r$ is a constructor term, and the conditions of $c$ are all pattern conditions. Note that pure-constructor rules are also *normal* [5]. A rewrite rule $l \to r \Leftarrow c$ is said to be of *type 3* [13] if $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l) \cup \mathcal{V}ar(c)$. A conditional rewrite rule $l \to r \Leftarrow u_1 \to v_1; \cdots ; u_n \to v_n$ of type 3 is called *deterministic* [9] if $\mathcal{V}ar(u_i) \subseteq \mathcal{V}ar(l, v_1, \ldots, v_{i-1})$ for all $i$ $(1 \le i \le n)$.

An *oriented conditional term rewriting system* (CTRS, for short) is a finite set $\mathcal{R}$ of oriented conditional rewrite rules. A *pure-constructor system* is a CTRS whose rewrite rules are all pure-constructor rewrite rules.[1] A CTRS $\mathcal{R}$ is called *deterministic* if all rewrite rules of $\mathcal{R}$ are deterministic.

▶ **Example 2.1.** CTRS $\mathcal{R}_{\mathsf{add}}$ in Section 1 is convertible to the following equivalent and deterministic pure-constructor system:

$$\mathcal{R}_{\mathsf{add}} = \left\{ \begin{array}{l} \mathsf{add}(0, y) \to y \\ \mathsf{add}(\mathsf{s}(x), y) \to \mathsf{s}(z) \Leftarrow \mathsf{add}(x, y) \to z \end{array} \right\}.$$

We introduce a relational logic over $\mathcal{T}(\mathcal{F}, \mathcal{V})$. An atom is a pair of terms $s$ and $t$, denoted by $s \to t$. Formulas are atoms, existentially quantified formulas, conjunction of formulas and implication of formulas. Satisfaction of a formula $\varphi$ by a pair of a relation $\rightharpoonup$ on $\mathcal{T}(\mathcal{F}, \mathcal{V})$ and a substitution $\sigma : \mathcal{V} \to \mathcal{T}(\mathcal{F}, \mathcal{V})$, denoted by $\langle \rightharpoonup, \sigma \rangle \models \varphi$, is inductively defined as follows:

- $\langle \rightharpoonup, \sigma \rangle \models u \to v$ iff $u\sigma \rightharpoonup v\sigma$,
- $\langle \rightharpoonup, \sigma \rangle \models \varphi \wedge \varphi'$ iff $\langle \rightharpoonup, \sigma \rangle \models \varphi$ and $\langle \rightharpoonup, \sigma \rangle \models \varphi'$, and
- $\langle \rightharpoonup, \sigma \rangle \models \varphi \Rightarrow \varphi'$ iff $\langle \rightharpoonup, \sigma \rangle \models \varphi$ implies $\langle \rightharpoonup, \sigma \rangle \models \varphi'$.

Note that a sequence of conditions $u_1 \to v_1; \cdots ; u_n \to v_n$ is regarded as conjunction $u_1 \to v_1 \wedge \cdots \wedge u_n \to v_n$. The *reflexive transitive closure* of a relation $\rightharpoonup$ is denoted by $\overset{*}{\rightharpoonup}$.

The $k$*-level reduction* $\to_{\mathcal{R}}^{(k)}$ of $\mathcal{R}$ is inductively defined as follows:

- $\to_{\mathcal{R}}^{(0)} = \emptyset$, and
- $\to_{\mathcal{R}}^{(j)} = \to_{\mathcal{R}}^{(j-1)} \cup \{(C[l\sigma], C[r\sigma]) \mid l \to r \Leftarrow c \in \mathcal{R},\ C \in \mathcal{T}(\mathcal{F} \cup \{\Box\}, \mathcal{V}),$
  $\sigma$ is a substitution, $\langle \overset{*}{\to}{}_{\mathcal{R}}^{(j-1)}, \sigma \rangle \models c\}$ for $j > 0$ .

The *reduction* $\to_{\mathcal{R}}$ is defined as $\bigcup_{k \ge 0} \to_{\mathcal{R}}^{(k)}$.

The constructor-based reduction $\underset{\mathsf{c}}{\to}_{\mathcal{R}}$ of a CTRS $\mathcal{R}$ [16] can be defined in the same way as the ordinary reduction of a CTRS except that matching substitutions are restricted to constructor substitutions. Note that $\underset{\mathsf{c}}{\to}_{\mathcal{R}} \subseteq \to_{\mathcal{R}}$.

## 3 Inverse Problem of Program Transformation

A program transformation is a procedure to generate another program from a given program and its main purpose is to improve execution efficiency of programs. On the other hand, the inverse of a program transformation is not a function, since program transformations are not one-to-one in general. The notion of the inverse of a program transformation is captured as an inverse image of a program under the transformation.

[1] The class of pure-constructor systems is the same as the class of normalized TRSs in [1].

**Unfold:**

$$\frac{\mathcal{R}_1 = \mathcal{R} \cup \{\rho :\ l \to r \Leftarrow c; u \to v; d\}}{\begin{array}{l}\mathcal{R}\ \cup \{l\sigma \to r\sigma \Leftarrow c\sigma; c'\sigma; d\sigma \mid \rho' :\ l' \to r' \Leftarrow c' \in \mathcal{R}_1, \\ \qquad\qquad \mathcal{V}ar(\rho) \cap \mathcal{V}ar(\rho') = \emptyset,\ \sigma = \mathrm{mgu}(\{u \sim l', v \sim r'\})\}\end{array}}$$

**Fold:** If $\rho' :\ l' \to r' \Leftarrow c' \in \mathcal{R}$,

$$\frac{\mathcal{R} \cup \{\rho :\ l \to r \Leftarrow c; c'\sigma; d\}}{\mathcal{R} \cup \{l \to r \Leftarrow c; l'\sigma \to r'\sigma; d\}}$$

Note that

1. transformations are performed by applying the transformation rules above in the direction from top to bottom,
2. each substitution $\sigma$ above is restricted to a constructor substitution,
3. each rewrite rule $\rho'$ above should be renamed so that $\mathcal{V}ar(\rho) \cap \mathcal{V}ar(\rho') = \emptyset$ holds, and
4. we sometimes say "Unfold $\rho$ by $\rho'$" and "Fold $\rho$ by $\rho'$" in applying the transformation rules above.

**Figure 1** Unfold/Fold Transformation Rules.

In this section, we formalize the inverse problem of an one-step program transformation by regarding it as a relation over programs. For example, an one-step transformation $T$ that converts a program $\mathcal{R}_1$ into $\mathcal{R}_2$ is a relation $T$ with $\mathcal{R}_1\ T\ \mathcal{R}_2$.

The inverse problem of an one-step program transformation is formalized as follows.

▶ **Definition 3.1.** Given a transformation $T$ and a program $\mathcal{R}_1$, the *inverse $T$ problem ($T^{-1}$ problem)* determines whether or not there exists a program $\mathcal{R}_0$ such that $\mathcal{R}_0\ T\ \mathcal{R}_1$ holds. If there exists such $\mathcal{R}_0$, we write $\mathcal{R}_1\ T^{-1}\ \mathcal{R}_0$, which means that transformation $T^{-1}$ is equals to inverse relation of $T$. $\mathcal{R}_0$ is called a *solution* of $T^{-1}$ problem for $\mathcal{R}_1$.

## 4 Inverse Unfold Problem

In the rest of the paper, we focus on Unfold$^{-1}$ problem, where we use Unfold/Fold transformation rules [14] on deterministic pure-constructor CTRSs. The definitions of those transformation rules are shown in Figure 1. Remark that a CTRS obtained by applying Unfold is equivalent to the original one, but a CTRS obtained by applying Fold, which is a derived rule of the one in [14], is not equivalent in general. More details on the correctness of Unfold/Fold transformations on pure-constructor systems are discussed in [14].

We revisit the examples in Section 1. $\mathcal{R}_{\mathsf{mult}}$ and $\mathcal{R}_{\mathsf{mult1}}$ in the following example are pure-constructor CTRSs equivalent to $\mathcal{R}_1$ and $\mathcal{R}_2$ in Section 1, respectively.

▶ **Example 4.1.** We obtain a CTRS $\mathcal{R}_{\mathsf{mult1}}$ by applying Unfold rule to the second rewrite rule of $\mathcal{R}_{\mathsf{mult}}$ using its own two rewrite rules.

$$\mathcal{R}_{\mathsf{mult}} = \left\{ \begin{array}{l} \mathsf{mult}(0, y) \to 0 \\ \mathsf{mult}(\mathsf{s}(x), y) \to v \Leftarrow \mathsf{mult}(x, y) \to z; \mathsf{add}(z, y) \to v \end{array} \right\}$$

$$\mathcal{R}_{\mathsf{mult1}} = \left\{ \begin{array}{l} \mathsf{mult}(0, y) \to 0 \\ \mathsf{mult}(\mathsf{s}(0), y) \to v \ \Leftarrow \mathsf{add}(0, y) \to v \\ \mathsf{mult}(\mathsf{s}^2(x), y) \to v \Leftarrow \mathsf{mult}(x, y) \to z; \mathsf{add}(z, y) \to w; \mathsf{add}(w, y) \to v \end{array} \right\}$$

Unfold$^{-1}$ problem can be regarded as an instance of Definition 3.1 i.e. a decision problem that determines, for a given CTRS $\mathcal{R}_1$, whether or not there exists a program $\mathcal{R}_0$, from which $\mathcal{R}_1$ is obtained by Unfold rule ($\mathcal{R}_0$ Unfold $\mathcal{R}_1$).

▶ **Example 4.2.** A solution of Unfold$^{-1}$ problem for CTRS $\mathcal{R}_{\mathsf{mult1}}$ is CTRS $\mathcal{R}_{\mathsf{mult}}$ because $\mathcal{R}_{\mathsf{mult}}$ Unfold $\mathcal{R}_{\mathsf{mult1}}$ holds in Example 4.1.

Program transformations are usually demanded to preserve the meaning of programs. In case of Example 4.1, the two functions mult defined in $\mathcal{R}_{\mathsf{mult}}$ and $\mathcal{R}_{\mathsf{mult1}}$ are required to derive the same normal forms if the same ground terms are given in the arguments. This preservation property is known as the combination of properties *Simulation soundness* and *simulation completeness*.

These properties for program transformation on pure-constructor systems are defined as follows [14].

▶ **Definition 4.3.** A transformation $T$ over CTRSs is *simulation sound* if and only if $s \xrightarrow[\mathsf{c}]{*}_{\mathcal{R}_2} t$ implies $s \xrightarrow[\mathsf{c}]{*}_{\mathcal{R}_1} t$ for any CTRSs $\mathcal{R}_1$, $\mathcal{R}_2$ such that $\mathcal{R}_1\ T\ \mathcal{R}_2$, and for any $s,t \in \mathcal{T}(\mathcal{D}_{\mathcal{R}_1} \cup \mathcal{C})$.

▶ **Definition 4.4.** A transformation $T$ over CTRSs is *simulation complete* if and only if $s \xrightarrow[\mathsf{c}]{*}_{\mathcal{R}_1} t$ implies $s \xrightarrow[\mathsf{c}]{*}_{\mathcal{R}_2} t$ for any CTRSs $\mathcal{R}_1$, $\mathcal{R}_2$ such that $\mathcal{R}_1\ T\ \mathcal{R}_2$, and for any $s,t \in \mathcal{T}(\mathcal{D}_{\mathcal{R}_1} \cup \mathcal{C})$.

In $T^{-1}$ problem, even if transformation $T$ is simulation sound and complete, $T^{-1}$ is not so in general. A sufficient condition for the simulation soundness and completeness of the inverse problem is easily shown as follows.

▶ **Theorem 4.5.** *If a transformation $T$ is simulation sound and complete, and $T$ introduce no new defined symbol, $T^{-1}$ is also simulation sound and complete.*

**Proof.** Let $\mathcal{R}_1$, $\mathcal{R}_2$ be CTRSs such that $\mathcal{R}_1\ T^{-1}\ \mathcal{R}_2$. Suppose $s,t \in \mathcal{T}(\mathcal{D}_{\mathcal{R}_1} \cup \mathcal{C})$. By the definition of $T^{-1}$ problem, $\mathcal{R}_2\ T\ \mathcal{R}_1$ holds. Since $T$ introduce no new defined symbol, i.e., $\mathcal{D}_{\mathcal{R}_1} = \mathcal{D}_{\mathcal{R}_2}$, it follows that $s,t \in \mathcal{T}(\mathcal{D}_{\mathcal{R}_2} \cup \mathcal{C})$. Combined this with $T$'s simulation soundness and completeness, ($s \xrightarrow[\mathsf{c}]{*}_{\mathcal{R}_1} t$ implies $s \xrightarrow[\mathsf{c}]{*}_{\mathcal{R}_2} t$) and ($s \xrightarrow[\mathsf{c}]{*}_{\mathcal{R}_2} t$ implies $s \xrightarrow[\mathsf{c}]{*}_{\mathcal{R}_1} t$) hold. ◀

Unfold rule in Figure 1 is simulation sound and complete [14], and introduces no new defined symbol. Thus, Theorem 4.5 derives the following corollary.

▶ **Corollary 4.6.** Unfold$^{-1}$ *is simulation sound and complete.*

This corollary guarantees both CTRSs before and after Unfold$^{-1}$ have the same rewrite relation.

## 5 Heuristics for Solving Inverse Unfold Problem

We propose a heuristic procedure for solving Unfold$^{-1}$ problems, which is shown in Figure 2. In this procedure, we generate a divergent sequence of rewrite rules by applying Unfold/Fold rules in Figure 1 from a given CTRS $\mathcal{R}_1$ (Step 1–4), generalize rules in the sequence by the difference matching [3] (Step 5) to obtain a solution candidate CTRS $\mathcal{R}_2$ for the Unfold$^{-1}$ problem. In Step 4 and 5, the simulation soundness and completeness are not necessarily preserved. Therefore, Step 6 is necessary in order to confirm that the obtained CTRS $\mathcal{R}_2$ is a solution, where $\mathcal{R}_2$ preserves the behavior of $\mathcal{R}_1$ by Corollary 4.6. Remark that in Step 2, "Unfolding" ($\mathcal{I}$)-labelled rule by ($b_0$)-, ($b_1$)- and ($\mathcal{I}$)-labelled rules yields ($\mathcal{I}b_0$)-, ($\mathcal{I}b_1$)- and ($\mathcal{I}^2$)-labelled rules, for example.

**Step 1** Give the labels $(b_0), (b_1), \ldots$ to each rule for a base case of the target function in the given CTRS $\mathcal{R}_1$ in order of argument size. Similarly, give the label $(\mathcal{I})$ to the rule for the induction case.
**Step 2** Unfold $(\mathcal{I}^n)$-labelled[1)] rule by all rules including itself. Attach each resulted rule the label obtained by concatenating the label of Unfolded rule and that of Unfolding rule in this order.
**Step 3** Again, do (Step 2) $l$ times[2)].
**Step 4** Fold each $(\mathcal{I}^n b_m)$-labelled rule by $(\mathcal{I}^n b_{m-1})$-labelled rule in lexicographically descending order of $(n, m)$. In an exceptional case, each $(\mathcal{I}^n b_0)$-labelled rule is "Fold"ed by $(\mathcal{I}^{n-1} b_{\mathsf{max}(m)})$-labelled rule. The each label of generated rules is $(\mathcal{I}^n b_m)'$, respectively.
**Step 5** Generalize rules in the divergent sequence generated in (Step 4) by the difference matching [3] to obtain a solution candidate CTRS $\mathcal{R}_2$.
**Step 6** If $\mathcal{R}_2$ Unfold $\mathcal{R}_1$ is satisfied, $\mathcal{R}_2$ is a solution of Unfold$^{-1}$ problem for $\mathcal{R}_1$.

1) $\mathcal{I}^n$ denotes $\overbrace{\mathcal{I} \cdots \mathcal{I}}^{n}$.
2) $l$ is a given fixed number.

**Figure 2** Heuristic procedure for Unfold$^{-1}$ problem.

▶ **Example 5.1.** We solve Unfold$^{-1}$ problem for $\mathcal{R}_{\mathsf{mult1}}$ in Example 4.1 by applying the heuristic procedure in Figure 2.

**Step 1:** First, we give labels to rewrite rules in $\mathcal{R}_{\mathsf{mult1}}$.

$$\mathcal{R}_{\mathsf{mult1}} = \left\{ \begin{array}{ll} \mathsf{mult}(0, y) \to 0 & (b_0) \\ \mathsf{mult}(\mathsf{s}(0), y) \to v \Leftarrow \mathsf{add}(0, y) \to v & (b_1) \\ \mathsf{mult}(\mathsf{s}^2(x), y) \to v \Leftarrow \mathsf{mult}(x, y) \to z; \mathsf{add}(z, y) \to w; \mathsf{add}(w, y) \to v & (\mathcal{I}) \end{array} \right\}$$

**Step 2:** Next, we "Unfold" $(\mathcal{I})$-labelled rule by $(b_0)$-, $(b_1)$- and $(\mathcal{I})$-labelled rules, which yields the following CTRS $\mathcal{R}_{\mathsf{mult2}}$.

$$\mathcal{R}_{\mathsf{mult2}} = \left\{ \begin{array}{ll} \mathsf{mult}(0, y) \to 0 & (b_0) \\ \mathsf{mult}(\mathsf{s}(0), y) \to v \Leftarrow \mathsf{add}(0, y) \to v & (b_1) \\ \mathsf{mult}(\mathsf{s}^2(0), y) \to v \Leftarrow \mathsf{add}(0, y) \to w; \mathsf{add}(w, y) \to v & (\mathcal{I}b_0) \\ \mathsf{mult}(\mathsf{s}^3(0), y) \to v \Leftarrow \mathsf{add}(0, y) \to w_2; \mathsf{add}(w_2, y) \to w; \mathsf{add}(w, y) \to v & (\mathcal{I}b_1) \\ \mathsf{mult}(\mathsf{s}^4(x), y) \to v \Leftarrow \mathsf{mult}(x, y) \to z; & \\ \qquad \mathsf{add}(z, y) \to w_3; \mathsf{add}(w_3, y) \to w_2; & \\ \qquad \mathsf{add}(w_2, y) \to w; \mathsf{add}(w, y) \to v & (\mathcal{I}^2) \end{array} \right\}$$

**Step 3:** Again, we "Unfold" $(\mathcal{I}^2)$-labelled rule in $\mathcal{R}_{\mathsf{mult2}}$ by the rules with the labels from $(b_0)$ through $(\mathcal{I}^2)$, which yields the following CTRS $\mathcal{R}_{\mathsf{mult3}}$.

$$\mathcal{R}_{\mathsf{mult3}} = \left\{ \begin{array}{ll} \mathsf{mult}(0, y) \to 0 & (b_0) \\ \mathsf{mult}(\mathsf{s}(0), y) \to v \Leftarrow \mathsf{add}(0, y) \to v & (b_1) \\ \mathsf{mult}(\mathsf{s}^2(0), y) \to v \Leftarrow \mathsf{add}(0, y) \to w; \mathsf{add}(w, y) \to v & (\mathcal{I}b_0) \\ \mathsf{mult}(\mathsf{s}^3(0), y) \to v \Leftarrow \mathsf{add}(0, y) \to w_2; \mathsf{add}(w_2, y) \to w; \mathsf{add}(w, y) \to v & (\mathcal{I}b_1) \\ \mathsf{mult}(\mathsf{s}^4(0), y) \to v \Leftarrow \mathsf{add}(0, y) \to w_3; \cdots; \mathsf{add}(w, y) \to v & (\mathcal{I}^2 b_0) \\ \mathsf{mult}(\mathsf{s}^5(0), y) \to v \Leftarrow \mathsf{add}(0, y) \to w_4; \cdots; \mathsf{add}(w, y) \to v & (\mathcal{I}^2 b_1) \\ \mathsf{mult}(\mathsf{s}^6(0), y) \to v \Leftarrow \mathsf{add}(0, y) \to w_5; \cdots; \mathsf{add}(w, y) \to v & (\mathcal{I}^3 b_0) \\ \mathsf{mult}(\mathsf{s}^7(0), y) \to v \Leftarrow \mathsf{add}(0, y) \to w_6; \cdots; \mathsf{add}(w, y) \to v & (\mathcal{I}^3 b_1) \\ \mathsf{mult}(\mathsf{s}^8(x), y) \to v \Leftarrow \mathsf{mult}(x, y) \to z; & \\ \qquad \mathsf{add}(z, y) \to w_7; \cdots; \mathsf{add}(w, y) \to v & (\mathcal{I}^4) \end{array} \right\}$$

**Table 1** Examples of heuristic procedure in Figure 2.

| Given CTRS | Solution CTRS |
|---|---|
| $\left\{\begin{array}{l}\mathsf{add}(0,y) \to y \\ \mathsf{add}(\mathsf{s}(0),y) \to \mathsf{s}(y) \\ \mathsf{add}(\mathsf{s}^2(x),y) \to \mathsf{s}^2(z) \Leftarrow \mathsf{add}(x,y) \to z\end{array}\right\}$ | – |
| $\left\{\begin{array}{l}\mathsf{mult}(0,y) \to 0 \\ \mathsf{mult}(\mathsf{s}(0),y) \to v \ \Leftarrow \mathsf{add}(0,y) \to v \\ \mathsf{mult}(\mathsf{s}^2(x),y) \to v \Leftarrow \mathsf{mult}(x,y) \to z; \\ \qquad\qquad\qquad\qquad \mathsf{add}(z,y) \to w; \\ \qquad\qquad\qquad\qquad \mathsf{add}(w,y) \to v\end{array}\right\}$ | $\left\{\begin{array}{l}\mathsf{mult}(0,y) \to 0 \\ \mathsf{mult}(\mathsf{s}(x),y) \to v \Leftarrow \mathsf{mult}(x,y) \to z; \\ \qquad\qquad\qquad\qquad \mathsf{add}(z,y) \to v\end{array}\right\}$ |
| $\left\{\begin{array}{l}\mathsf{rev}([\,]) \to [\,] \\ \mathsf{rev}(x_1 : [\,]) \to zs \Leftarrow \mathsf{app}([\,], x_1 : [\,]) \to zs \\ \mathsf{rev}(x_1 : x_2 : xs) \to zs \\ \qquad\qquad \Leftarrow \mathsf{rev}(xs) \to zs_2; \\ \qquad\qquad\quad \mathsf{app}(zs_2, x_2 : [\,]) \to zs_1; \\ \qquad\qquad\quad \mathsf{app}(zs_1, x_1 : [\,]) \to zs\end{array}\right\}$ | $\left\{\begin{array}{l}\mathsf{rev}([\,]) \to [\,] \\ \mathsf{rev}(x : xs) \to zs \Leftarrow \mathsf{rev}(xs) \to zs_1; \\ \qquad\qquad\qquad\qquad \mathsf{app}(zs_1, x : [\,]) \to zs\end{array}\right\}$ |
| $\left\{\begin{array}{l}\mathsf{frev}([\,], ys) \to ys \\ \mathsf{frev}(x : [\,], ys) \to x : ys \\ \mathsf{frev}(x_1 : x_2 : xs, ys) \to zs \\ \qquad\qquad \Leftarrow \mathsf{frev}(xs, x_2 : x_1 : ys) \to zs\end{array}\right\}$ | – |

**Step 4:** We "Fold" ($\mathcal{I}^3 b_1$)-labelled rule by ($\mathcal{I}^3 b_0$)-labelled rule, which yields a rule "$\mathsf{mult}(\mathsf{s}^7(0),y) \to v \Leftarrow \mathsf{mult}(\mathsf{s}^6(0),y) \to w, \mathsf{add}(w,y) \to v$". Similarly, the rules with the labels from ($\mathcal{I}^3 b_0$) through ($\mathcal{I} b_0$) are "Folded" by the rules with the labels from ($\mathcal{I}^2 b_1$) through ($b_1$), respectively. We get the following CTRS $\mathcal{R}_{\mathsf{mult3'}}$.

$$\mathcal{R}_{\mathsf{mult3'}} = \left\{\begin{array}{lll}
\mathsf{mult}(0,y) \to 0 & & (b_0) \\
\mathsf{mult}(\mathsf{s}(0),y) \to v & \Leftarrow \mathsf{add}(0,y) \to v & (b_1) \\
\mathsf{mult}(\mathsf{s}(\mathsf{s}(0)),y) \to v & \Leftarrow \mathsf{mult}(\mathsf{s}(0),y) \to w; \mathsf{add}(w,y) \to v & (\mathcal{I}b_0)' \\
\mathsf{mult}(\mathsf{s}(\mathsf{s}^2(0)),y) \to v & \Leftarrow \mathsf{mult}(\mathsf{s}^2(0),y) \to w; \mathsf{add}(w,y) \to v & (\mathcal{I}b_1)' \\
\mathsf{mult}(\mathsf{s}(\mathsf{s}^3(0)),y) \to v & \Leftarrow \mathsf{mult}(\mathsf{s}^3(0),y) \to w; \mathsf{add}(w,y) \to v & (\mathcal{I}^2b_0)' \\
\mathsf{mult}(\mathsf{s}(\mathsf{s}^4(0)),y) \to v & \Leftarrow \mathsf{mult}(\mathsf{s}^4(0),y) \to w; \mathsf{add}(w,y) \to v & (\mathcal{I}^2b_1)' \\
\mathsf{mult}(\mathsf{s}(\mathsf{s}^5(0)),y) \to v & \Leftarrow \mathsf{mult}(\mathsf{s}^5(0),y) \to w; \mathsf{add}(w,y) \to v & (\mathcal{I}^3b_0)' \\
\mathsf{mult}(\mathsf{s}(\mathsf{s}^6(0)),y) \to v & \Leftarrow \mathsf{mult}(\mathsf{s}^6(0),y) \to w; \mathsf{add}(w,y) \to v & (\mathcal{I}^3b_1)' \\
\mathsf{mult}(\mathsf{s}^8(x),y) \to v & \Leftarrow \mathsf{mult}(x,y) \to z; & \\
 & \quad \mathsf{add}(z,y) \to w_7; \cdots ; \mathsf{add}(w,y) \to v & (\mathcal{I}^4)
\end{array}\right\}$$

**Step 5:** Generalize the divergent rules in $\mathcal{R}_{\mathsf{mult3'}}$ with the labels from ($\mathcal{I}b_0$)$'$ through ($\mathcal{I}^3 b_1$)$'$ by the difference matching, which yields a rule "$\mathsf{mult}(\mathsf{s}(x),y) \to v \Leftarrow \mathsf{mult}(x,y) \to w, \mathsf{add}(w,y) \to v$". Now $\mathcal{R}_{\mathsf{mult}}$ is obtained as a solution candidate of the Unfold$^{-1}$ problem for $\mathcal{R}_{\mathsf{mult1}}$.

**Step 6:** It is confirmed that $\mathcal{R}_{\mathsf{mult}}$ Unfold $\mathcal{R}_{\mathsf{mult1}}$. Thus $\mathcal{R}_{\mathsf{mult}}$ is certainly a solution.

Table 1 shows the results obtained by applying the procedure in Figure 2 by hand to four problems: addition of two natural numbers, multiplication of two natural numbers, reverse of a list and fast reverse of a list. Here, '–' in the table shows that the procedure failed to solve Unfold$^{-1}$ for the problem.

## 6 Heuristics Introducing Identity Function

Consider the first CTRS $\mathcal{R}_{\mathsf{add1}}$ in Table 1, for which the procedure in Section 5 fails.

$$\mathcal{R}_{\mathsf{add1}} = \left\{ \begin{array}{l} \mathsf{add}(0, y) \to y \\ \mathsf{add}(\mathsf{s}(0), y) \to \mathsf{s}(y) \\ \mathsf{add}(\mathsf{s}^2(x), y) \to \mathsf{s}^2(z) \Leftarrow \mathsf{add}(x, y) \to z \end{array} \right\}.$$

The following CTRS is obtained from $\mathcal{R}_{\mathsf{add1}}$ as an intermediate result by the heuristic procedure in Figure 2; applying Step 1 to Step 3 (Step 3 twice):

$$\mathcal{R}' = \left\{ \begin{array}{ll} \mathsf{add}(0, y) \to y & (b_0) \\ \mathsf{add}(\mathsf{s}(0), y) \to \mathsf{s}(y) & (b_1) \\ \mathsf{add}(\mathsf{s}^2(0), y) \to \mathsf{s}^2(y) & (\mathcal{I}b_0) \\ \mathsf{add}(\mathsf{s}^3(0), y) \to \mathsf{s}^3(y) & (\mathcal{I}b_1) \\ \mathsf{add}(\mathsf{s}^4(0), y) \to \mathsf{s}^4(y) & (\mathcal{I}^2b_0) \\ \mathsf{add}(\mathsf{s}^5(0), y) \to \mathsf{s}^5(y) & (\mathcal{I}^2b_1) \\ \mathsf{add}(\mathsf{s}^6(0), y) \to \mathsf{s}^6(y) & (\mathcal{I}^3b_0) \\ \mathsf{add}(\mathsf{s}^7(0), y) \to \mathsf{s}^7(y) & (\mathcal{I}^3b_1) \\ \mathsf{add}(\mathsf{s}^8(x), y) \to \mathsf{s}^8(z) \Leftarrow \mathsf{add}(x, y) \to z & (\mathcal{I}^4) \end{array} \right\}.$$

Then any applications of Fold rule in Step 4 are impossible, because all rules with the labels from $(b_0)$ through $(\mathcal{I}^3b_1)$ have no conditional part, which are necessary in applying Fold rule. If the second rule of $\mathcal{R}_{\mathsf{add1}}$ were of the form

$$\mathsf{add}(\mathsf{s}(0), y) \to z \Leftarrow \mathsf{add}(0, y) \to z,$$

then the transformation would be successful. Since the former is obtained by simplifying the latter, this means that some necessary information in the conditional part may be lost by a simplification. One possibility to avoid this issue is recovering the information from the simpler rule such as the former. It is, however, difficult to find a clue. Instead of recovering the conditional part directly, we adopt an alternative that inserts a condition with transparent function $\mathsf{id}$, which is identity function defined by

$$\mathsf{id}(x) \to x.$$

We introduce a conditional part with the identity function to make each right-hand side of body parts is a variable. The CTRS $\mathcal{R}_{\mathsf{add1}}$ is transformed into the following CTRS:

$$\mathcal{R}_{\mathsf{add1}'} = \left\{ \begin{array}{l} \mathsf{add}(0, y) \to y \\ \mathsf{add}(\mathsf{s}(0), y) \to w \ \Leftarrow \mathsf{id}(\mathsf{s}(y)) \to w \\ \mathsf{add}(\mathsf{s}^2(x), y) \to w \Leftarrow \mathsf{add}(x, y) \to z; \mathsf{id}(\mathsf{s}(z)) \to w_1; \mathsf{id}(\mathsf{s}(w_1)) \to w \end{array} \right\}.$$

This process is formalized as the following procedure.

▶ **Procedure 6.1** (id attachment)**.** The id-attached rule of a pure-constructor rule $\rho : l \to r \Leftarrow c$ is constructed as follows.

1. If $r \notin \mathcal{V}$, then convert $\rho$ into $\rho' : l \to z \Leftarrow c; \mathsf{id}(r) \to z$. Otherwise, let $\rho'$ be $\rho$ itself.
2. Replace a condition in $\rho'$ of the form $\mathsf{id}(g(t_1, \ldots, t_i, \ldots, t_n)) \to v$ such that $t_i \notin \mathcal{V}$ with $\mathsf{id}(t_i) \to z_i; \mathsf{id}(g(t_1, \ldots, z_i, \ldots, t_n)) \to v$

**Step 0** Apply Procedure 6.1 to each rule in the given CTRS $\mathcal{R}_1$.
**Step 1** Give labels to each rule in the same way as Figure 2 (Step 1).
**Step 2** Unfold rules in the same way as Figure 2 (Step 2). For each "Unfolded" rule, apply Procedure 6.1 to the generated rule.
**Step 3–6** Do each step in the same way as Figure 2.

**Figure 3** Modified heuristic procedure for Unfold$^{-1}$ problem.

**3.** Repeat 2 until it can not be applicable.
Note that variables $z$ and $z_i$ above must be fresh.

Actually, this procedure must be applied after each Unfolding application during Step 2–3 in Figure 2 because conditions of the generated rule may disappear by Unfolding. The modified heuristics we propose is summarized in Figure 3.

▶ **Example 6.2.** We solve Unfold$^{-1}$ problem for $\mathcal{R}_{\mathsf{add1}}$ applying the modified heuristic procedure.
**Step 0:** As described above, we obtain $\mathcal{R}_{\mathsf{add1'}}$ by Step 0.
**Step 1–3:** As a result of Step 1–3, we obtain

$$\mathcal{R}_{\mathsf{add2}} = \left\{ \begin{array}{lll}
\mathsf{add}(0,y) \to y & & (b_0) \\
\mathsf{add}(\mathsf{s}(0),y) \to w & \Leftarrow \mathsf{id}(\mathsf{s}(y)) \to w & (b_1) \\
\mathsf{add}(\mathsf{s}^2(0),y) \to w & \Leftarrow \mathsf{id}(\mathsf{s}(y)) \to w_1; \mathsf{id}(\mathsf{s}(w_1)) \to w & (\mathcal{I}b_0) \\
\mathsf{add}(\mathsf{s}^3(0),y) \to w & \Leftarrow \mathsf{id}(\mathsf{s}(y)) \to w_2; \mathsf{id}(\mathsf{s}(w_2)) \to w_1; \mathsf{id}(\mathsf{s}(w_1)) \to w & (\mathcal{I}b_1) \\
\mathsf{add}(\mathsf{s}^4(0),y) \to w & \Leftarrow \mathsf{id}(\mathsf{s}(y)) \to w_3; \mathsf{id}(\mathsf{s}(w_3)) \to w_2; \cdots; \mathsf{id}(\mathsf{s}(w_1)) \to w & (\mathcal{I}^2b_0) \\
\mathsf{add}(\mathsf{s}^5(0),y) \to w & \Leftarrow \mathsf{id}(\mathsf{s}(y)) \to w_4; \mathsf{id}(\mathsf{s}(w_4)) \to w_3; \cdots; \mathsf{id}(\mathsf{s}(w_1)) \to w & (\mathcal{I}^2b_1) \\
\mathsf{add}(\mathsf{s}^6(0),y) \to w & \Leftarrow \mathsf{id}(\mathsf{s}(y)) \to w_5; \mathsf{id}(\mathsf{s}(w_5)) \to w_4; \cdots; \mathsf{id}(\mathsf{s}(w_1)) \to w & (\mathcal{I}^3b_0) \\
\mathsf{add}(\mathsf{s}^7(0),y) \to w & \Leftarrow \mathsf{id}(\mathsf{s}(y)) \to w_6; \mathsf{id}(\mathsf{s}(w_6)) \to w_5; \cdots; \mathsf{id}(\mathsf{s}(w_1)) \to w & (\mathcal{I}^3b_1) \\
\mathsf{add}(\mathsf{s}^8(x),y) \to w & \Leftarrow \mathsf{add}(x,y) \to z; \mathsf{id}(\mathsf{s}(z)) \to w_7; \cdots; \mathsf{id}(\mathsf{s}(w_1)) \to w; & (\mathcal{I}^4)
\end{array} \right\}.$$

**Step 4:** Fold transformations in Step 4 create the following rules.

$$\mathcal{R}_{\mathsf{add2'}} = \left\{ \begin{array}{lll}
\mathsf{add}(0,y) \to y & & (b_0) \\
\mathsf{add}(\mathsf{s}(0),y) \to w & \Leftarrow \mathsf{id}(\mathsf{s}(y)) \to w & (b_1) \\
\mathsf{add}(\mathsf{s}(\mathsf{s}(0)),y) \to w & \Leftarrow \mathsf{add}(\mathsf{s}(0),y) \to w_1; \mathsf{id}(\mathsf{s}(w_1)) \to w & (\mathcal{I}b_0)' \\
\mathsf{add}(\mathsf{s}(\mathsf{s}^2(0)),y) \to w & \Leftarrow \mathsf{add}(\mathsf{s}^2(0),y) \to w_1; \mathsf{id}(\mathsf{s}(w_1)) \to w; & (\mathcal{I}b_1)' \\
\mathsf{add}(\mathsf{s}(\mathsf{s}^3(0)),y) \to w & \Leftarrow \mathsf{add}(\mathsf{s}^3(0),y) \to w_1; \mathsf{id}(\mathsf{s}(w_1)) \to w & (\mathcal{I}^2b_0)' \\
\mathsf{add}(\mathsf{s}(\mathsf{s}^4(0)),y) \to w & \Leftarrow \mathsf{add}(\mathsf{s}^4(0),y) \to w_1; \mathsf{id}(\mathsf{s}(w_1)) \to w & (\mathcal{I}^2b_1)' \\
\mathsf{add}(\mathsf{s}(\mathsf{s}^5(0)),y) \to w & \Leftarrow \mathsf{add}(\mathsf{s}^5(0),y) \to w_1; \mathsf{id}(\mathsf{s}(w_1)) \to w & (\mathcal{I}^3b_0)' \\
\mathsf{add}(\mathsf{s}(\mathsf{s}^6(0)),y) \to w & \Leftarrow \mathsf{add}(\mathsf{s}^6(0),y) \to w_1; \mathsf{id}(\mathsf{s}(w_1)) \to w & (\mathcal{I}^3b_1)' \\
\mathsf{add}(\mathsf{s}^8(x),y) \to w & \Leftarrow \mathsf{add}(x,y) \to z; \mathsf{id}(\mathsf{s}(z)) \to w_7; \cdots; & \\
 & \qquad\qquad \mathsf{id}(\mathsf{s}(w_1)) \to w; & (\mathcal{I}^4)
\end{array} \right\}.$$

**Step 5:** Generalize the divergent rules in $\mathcal{R}_{\mathsf{add2'}}$ with the labels from $(\mathcal{I}b_0)'$ through $(\mathcal{I}^3b_1)'$ by the difference matching, which yields a rule "$\mathsf{add}(\mathsf{s}(x),y) \to w \Leftarrow \mathsf{add}(x,y) \to w_1; \mathsf{id}(\mathsf{s}(w_1)) \to w$", which is equal to "$\mathsf{add}(\mathsf{s}(x),y) \to \mathsf{s}(w_1) \Leftarrow \mathsf{add}(x,y) \to w_1$". So a solution candidate of

**Table 2** Examples of the modified heuristic procedure in Figure 3.

| Given CTRS | Solution CTRS |
|---|---|
| $\left\{\begin{array}{l}\mathsf{add}(0,y) \to y\\ \mathsf{add}(\mathsf{s}(0),y) \to \mathsf{s}(y)\\ \mathsf{add}(\mathsf{s}^2(x),y) \to \mathsf{s}^2(z) \Leftarrow \mathsf{add}(x,y) \to z\end{array}\right\}$ | $\left\{\begin{array}{l}\mathsf{add}(0,y) \to y\\ \mathsf{add}(\mathsf{s}(x),y) \to \mathsf{s}(z) \Leftarrow \mathsf{add}(x,y) \to z\end{array}\right\}$ |
| $\left\{\begin{array}{l}\mathsf{frev}([\,],ys) \to ys\\ \mathsf{frev}(x:[\,],ys) \to x:ys\\ \mathsf{frev}(x_1:x_2:xs,ys) \to zs\\ \qquad\qquad \Leftarrow \mathsf{frev}(xs,x_2:x_1:ys) \to zs\end{array}\right\}$ | $\left\{\begin{array}{l}\mathsf{frev}([\,],ys) \to ys\\ \mathsf{frev}(x:xs,ys) \to zs\\ \qquad\qquad \Leftarrow \mathsf{frev}(xs,x:ys) \to zs\end{array}\right\}$ |

Unfold$^{-1}$ problem for $\mathcal{R}_{\mathsf{add1}}$ is

$$\mathcal{R}_{\mathsf{add}} = \left\{\begin{array}{l}\mathsf{add}(0,y) \to y\\ \mathsf{add}(\mathsf{s}(x),y) \to \mathsf{s}(z) \Leftarrow \mathsf{add}(x,y) \to z\end{array}\right\}.$$

**Step 6:** It is confirmed that $\mathcal{R}_{\mathsf{add}}$ Unfold $\mathcal{R}_{\mathsf{add1}}$. Thus $\mathcal{R}_{\mathsf{add}}$ is certainly a solution.

Table 2 shows the results by applying the modified heuristics to the failed problems in Table 1.

## 7 Application

In this section, we show an example induced from the program inversion[10, 15, 16, 17, 18]. Consider the following TRS $\mathcal{R}_{\mathsf{rev}'}$, which defines a fast reverse function of a list:

$$\mathcal{R}_{\mathsf{rev}'} = \left\{\begin{array}{l}\mathsf{reverse}(xs) \to \mathsf{frev}(xs,[\,])\\ \mathsf{frev}([\,],ys) \to ys\\ \mathsf{frev}(x:[\,],ys) \to x:ys\\ \mathsf{frev}(x_1:x_2:xs,ys) \to \mathsf{frev}(xs,x_2:x_1:ys)\end{array}\right\}.$$

Note that the definition of frev is convertible to an equivalent pure-constructor CTRS in the second column of Table 2. For TRS $\mathcal{R}_{\mathsf{rev}'}$, a program inversion tool *repius* [2] produces the following CTRS:

$$\mathcal{R}_{\mathsf{invrev}'} = \left\{\begin{array}{l}\mathsf{inv\text{-}reverse}(ys) \to \mathsf{tp1}(xs) \Leftarrow \mathsf{tinv\text{-}frev}([\,],ys) \to \mathsf{tp2}(xs,[\,]),\\ \mathsf{inv\text{-}reverse}(x:ys) \to \mathsf{tp1}(xs) \Leftarrow \mathsf{tinv\text{-}frev}(x:[\,],ys) \to \mathsf{tp2}(xs,[\,]),\\ \mathsf{tinv\text{-}frev}(xs,[\,]) \to \mathsf{tp2}(xs,[\,]),\\ \mathsf{tinv\text{-}frev}(xs,x_2:x_1:ys) \to \mathsf{tinv\text{-}frev}(x_1:x_2:xs,ys).\end{array}\right\},$$

where $\mathsf{tp1}(\cdot)$ and $\mathsf{tp2}(\cdot,\cdot)$ are constructors introduced by *repius* for representing 1-tuple and 2-tuple, respectively. The function inv-reverse in $\mathcal{R}_{\mathsf{invrev}'}$ works as the inversion of frev, but non-determinacy in computation is necessary to obtain the expected results; the first rule should be applied to an odd-length list and the second rule to even-length list.

Next, we consider the following definition of frev, which is the solution CTRS in Table 2.

$$\mathcal{R}_{\mathsf{rev}} = \left\{\begin{array}{l}\mathsf{reverse}(xs) \to \mathsf{frev}(xs,[\,]),\\ \mathsf{frev}([\,],ys) \to ys,\\ \mathsf{frev}(x:xs,ys) \to \mathsf{frev}(xs,x:ys)\end{array}\right\}.$$

[2] `http://www.trs.cm.is.nagoya-u.ac.jp/repius/`

For this TRS, *repius* produces the following CTRS:

$$\mathcal{R}_{\mathsf{invrev}} = \left\{ \begin{array}{l} \mathsf{inv\text{-}reverse}(ys) \to \mathsf{tp1}(xs) \Leftarrow \mathsf{tinv\text{-}frev}([\,], ys) \to \mathsf{tp2}(xs, [\,]) \\ \mathsf{tinv\text{-}frev}(xs, [\,]) \to \mathsf{tp2}(xs, [\,]) \\ \mathsf{tinv\text{-}frev}(xs, x : ys) \to \mathsf{tinv\text{-}frev}(x : xs, ys) \end{array} \right\}.$$

The CTRS $\mathcal{R}_{\mathsf{invrev}}$ is left-linear and non-overlapping and hence non-determinacy is not necessary any more.

## 8 Conclusion

In this paper, we formalized the inverse problem of an one-step program transformation, and focused on inverse Unfold problem, which is simulation sound and complete. For this problem, we proposed a heuristic procedure and its improvement with the identity function. Using these heuristics, we have also shown some successful examples and an application example on program inversion.

As mentioned in Section 1, we used pure-constructor systems as a platform because of the firmness of the structure of the rules. However, the heuristics proposed in this paper may be modified for the general TRSs and unfoldings for them. Moreover, in that framework, $\mathsf{id}$ symbol in Section 6 might not be necessary. It is also interesting to consider this issue.

We should address the following future tasks.

**Target:** So far, the scope of heuristic solvings in this paper is limited to functions whose arguments consist of simple list-like data structures. Tree-like data structures and mutual recursive functions will be considered as targets. We should open the class of CTRSs in which our heuristics succeeds.

**Completeness:** We will find a subclass for which the heuristic procedure is complete; the procedure can always find a solution if it exists.

**Mechanization:** In our heuristics, there are multiple choices which rules to be unfolded/folded. Strategies to narrow down the options for automation is promising.

**Acknowledgements.** We would like to thank the anonymous referees for their corrections and valuable comments.

## References

1 Jesús Manuel Almendros-Jiménez and Germán Vidal. Automatic partial inversion of inductively sequential functions. In Zoltán Horváth, Viktória Zsók, and Andrew Butterfield, editors, *Proc. of 18th International Symposium on Implementation and Application of Functional Languages*, volume 4449 of *Lecture Notes in Computer Science*, pages 253–270, 2007.

2 Franz Baader and Tobias Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

3 David Basin and Toby Walsh. Difference matching. In Deepak Kapur, editor, *11th International Conference on Automated Deduction*, volume 607 of *Lecture Notes in Computer Science*, pages 295–309. Springer, 1992.

4 N. Bensaou and Irène Guessarian. Transforming constraint logic programs. *Theoretical Computer Science*, 206(1-2):81–125, 1998.

5 Jan A. Bergstra and Jan Willem Klop. Conditional rewrite rules: Confluence and termination. *Journal of Computer and System Sciences*, 32(3):323–362, 1986.

6 Rod M. Burstall and John Darlington. A transformation system for developing recursive programs. *Journal of the ACM*, 24(1):44–67, 1977.

7 Sandro Etalle and Maurizio Gabbrielli. Transformations of CLP modules. *Theoretical Computer Science*, 166(1-2):101–146, 1996.

8 Fabio Fioravanti, Alberto Pettorossi, and Maurizio Proietti. Transformation rules for locally stratified constraint logic programs. In Maurice Bruynooghe and Kung-Kiu Lau, editors, *Program Development in Computational Logic*, volume 3049 of *Lecture Notes in Computer Science*, pages 77–89. Springer, 2004.

9 Harald Ganzinger. Order-sorted completion: The many-sorted way. *Theoretical Computer Science*, 89(1):3–32, 1991.

10 Robert Glück and Masahiko Kawabe. A method for automatic program inversion based on LR(0) parsing. *Fundamenta Informmaticae*, 66(4):367–395, 2005.

11 Tadashi Kanamori and Kenji Horiuchi. Construction of logic programs based on generalized unfold/fold rules. In Jean-Louis Lassez, editor, *Proceedings of the 4th International Conference on Logic Programming*, pages 744–768, 1987.

12 Michael J. Maher. A transformation system for deductive database modules with perfect model semantics. *Theoretical Computer Science*, 110(2):377–403, 1993.

13 Aart Middeldorp and Erik Hamoen. Completeness results for basic narrowing. *Applicable Algebra in Engineering, Communication and Computing*, 5:213–253, 1994.

14 Masanori Nagashima, Masahiko Sakai, and Toshiki Sakabe. Determinization of conditional term rewriting systems. *Theoretical Computer Science*, 464:72–89, 2012.

15 Naoki Nishida, Masahiko Sakai, and Toshiki Sakabe. Partial inversion of constructor term rewriting systems. In Jürgen Giesl, editor, *Proc. of the 16th Int'l Conf. on Rewriting Techniques and Applications*, volume 3467 of *LNCS*, pages 264–278. Springer, 2005.

16 Naoki Nishida and Germán Vidal. Program inversion for tail recursive functions. In Manfred Schmidt-Schauß, editor, *Proceedings of the 22nd International Conference on Rewriting Techniques and Applications*, volume 10 of *LIPIcs*, pages 283–298. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2011.

17 Naoki Nishida and Germán Vidal. Computing more specific versions of conditional rewriting systems. In Elvira Albert, editor, *Revised Selected Papers of the 22nd International Symposium on Logic-Based*, volume 7844 of *Lecture Notes in Computer Science*, pages 137–154, 2013.

18 Minami Niwa, Naoki Nishida, and Masahiko Sakai. Extending matching operation in grammar program for program inversion. In Elvira Albert, editor, *Informal Proceedings of the 22nd International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR 2012)*, pages 130–139, 2012.

19 Enno Ohlebusch. *Advanced Topics in Term Rewriting.* Springer-Verlag, 2002.

20 Alberto Pettorossi, Maurizio Proietti, and Valerio Senni. Constraint-based correctness proofs for logic program transformations. *Formal Aspects of Computing*, 24(4–6):569–594, 2012.

21 Abhik Roychoudhury, K. Narayan Kumar, C. R. Ramakrishnan, and I. V. Ramakrishnan. Beyond Tamaki-Sato style unfold/fold transformations for normal logic programs. *International Journal of Foundations of Computer Science*, 13(3):387–403, 2002.

22 Abhik Roychoudhury, K. Narayan Kumar, C. R. Ramakrishnan, and I. V. Ramakrishnan. An unfold/fold transformation framework for definite logic programs. *ACM Transactions on Programming Languages and Systems*, 26(3):464–509, 2004.

23 David Sands. Total correctness by local improvement in the transformation of functional programs. *ACM Trans. on Programming Languages and Systems*, 18(2):175–234, 1996.

24 Taisuke Sato. Equivalence-preserving first-order unfold/fold transformation systems. *Theoretical Computer Science*, 105(1):57–84, 1992.

25 Hirohisa Seki. Unfold/fold transformation of general logic programs for the well-founded semantics. *Journal of Logic Programming*, 16(1):5–23, 1993.