Correctness of Context-Moving Transformations for Term Rewriting Systems

Koichi Sato, Kentaro Kikuchi, Takahito Aoto, and Yoshihito Toyama

RIEC, Tohoku University
2-1-1 Katahira, Aoba-ku, Sendai, Miyagi, 980-8577, Japan
{ koichi, kentaro, aoto, toyama }@nue.riec.tohoku.ac.jp

Abstract. Proofs by induction are often incompatible with functions in tail-recursive form as the accumulator changes in the course of unfolding the definitions. Context-moving and context-splitting (Giesl, 2000) for functional programs transform tail-recursive programs into non tail-recursive ones which are more suitable for proofs by induction and thus for verification. In this paper, we formulate context-moving and context-splitting transformations in the framework of term rewriting systems, and prove their correctness with respect to both eager evaluation semantics and initial algebra semantics under some conditions on the programs to be transformed. The conditions for the correctness with respect to initial algebra semantics can be checked by automated methods for inductive theorem proving developed in the field of term rewriting systems.

Key words: Tail-Recursion, Program Transformation, Term Rewriting System, Inductive Theorem Proving

1 Introduction

Proofs by induction are fundamental in software verification and thus dealt with by many automated theorem provers. An *inductive theorem* of a *term rewriting system* (*TRS* for short) is an equation valid in the initial algebra of the TRS. Inductive theorems correspond to the equations that can be shown by induction on the data structures, and various automated methods have been investigated for proving inductive validity of TRSs [1–3, 7, 9, 10].

Recursive definition is a fundamental tool in various areas. A recursive definition of a function in which the body of the definition is a recursive call of the function itself (with different arguments, typically) is called *tail-recursive*. When evaluating a function call, if the function definition is given in a tail-recursive form, the environment of the function call does not need to be kept to deal with further computations that manipulate the results of its recursive calls. Thus, programs in which function definitions are given in tail-recursive forms are compiled into codes removing extra overheads in function calls. Thus, tail-recursive programs attain both efficiency and readability. However, proofs by induction are often incompatible with tail-recursive definitions, as can be seen in the following example.

Example 1 (tail-recursion and proofs by induction). Let us consider the following rewrite rules in tail-recursive form computing the addition of natural numbers:

$$R = \{\ Add(0, y) \to y,\ \ Add(S(x), y) \to Add(x, S(y))\ \}$$

Let us consider proving $Add(x, 0) \doteq x$ by induction on x. In the induction step where $x = S(x')$, one needs to show the equation $Add(x', S(0)) \doteq S(x')$ obtained by unfolding the equation. However, one cannot apply the induction hypothesis $Add(x', 0) \doteq x'$ to this equation, since the second argument is different.

As the second argument y in the rewrite rules in Example 1, a tail-recursive definition usually contains a variable called an *accumulator* which keeps intermediate results of the computation and is passed to the return value at the final recursive call. By unfolding the definition, the value of the accumulator changes step by step in the course of the computation; in proofs by induction, this change of the value makes the application of the induction hypothesis impossible. In this way, proofs by induction are often incompatible with tail-recursive definitions. Most methods for proving inductive theorems of TRSs containing tail-recursive rules (tail-recursive TRSs) suffer a similar difficulty.

On the other hand, "simple" recursive definitions do not suffer such a problem. For example, a "simple" version of the TRS for addition would be the following usual definition.

Example 2 (simple recursion and proofs by induction). Let R' be the following TRS.

$$R' = \{\ Add(0, y) \to y,\ \ Add(S(x), y) \to S(Add(x, y))\ \}$$

Now let us prove the same equation $Add(x, 0) \doteq x$ of Example 1 using R'. The base step is trivial, and in the induction step, one obtains an equation $S(Add(x', 0)) \doteq S(x')$ by unfolding the definition. This time, one can apply the induction hypothesis $Add(x', 0) \doteq x'$, and thus the proof succeeds.

The TRS R' of Example 2 can be obtained from the TRS R of Example 1 by transforming the rhs of the second rule from $Add(x, S(y))$ to $S(Add(x, y))$, i.e., transforming the rhs of the rewrite rule in such a way that the context $S(\Box)$ around the accumulator y is moved outside of the recursive call $Add(x, y)$. Generalizing such a transformation, J. Giesl [4] proposed *context-moving* and *context-splitting* transformations for a particular form of functional programs with eager evaluation. These transformations, under some conditions, transform tail-recursive programs into equivalent "simple" recursive programs more suitable for theorem proving employing proofs by induction.

In a previous paper [8], we formulated context-moving and context-splitting transformations *for TRSs*, and showed their correctness in the case where input TRSs are orthogonal. We also proposed an approach for inductive theorem proving which combines these transformations with *rewriting induction* [7]. It was demonstrated by experiments that the approach is effective for proving inductive theorems of tail-recursive TRSs, compared to other systems based on rewriting induction [3, 9, 10] (the system of [9] is equipped with lemma generation techniques in [2, 11, 12]).

In the present paper, we focus on the correctness of the context-moving and context-splitting transformations as formulated in [8] but where input TRSs are more general than orthogonal. To clarify the difference to the approach in [4], we also show the correctness of the context-moving transformation where input and output TRSs are evaluated by a deterministic eager strategy and thus can be seen as a faithful representation of the functional programs discussed in [4].

The sufficient condition of the context-moving transformation for TRSs with eager evaluation is identical to that of [4]. The condition is based on whether two terms are evaluated to the same value by the input TRS (cf. Definition 2). However, this notion does not necessarily coincide with equality in the initial algebra of the TRS, and so is not an inductive theorem in the traditional sense. Hence, the condition cannot in general be verified by automated methods for proving inductive theorems as developed in the field of TRSs. This is an obstacle to implementing our approach to proving inductive theorems of tail-recursive TRSs.

On the other hand, the sufficient conditions of the context-moving and context-splitting transformations in the present paper are precisely equality in the initial algebra, and so can be checked by an inductive theorem prover. Moreover, as consequences of the correctness under the conditions, it turns out that the context-moving and context-splitting transformations preserve equality in the initial algebra, and the terms in each equivalence class have the same normal form with respect to rewriting by the TRSs before and after the transformations.

The contributions of the paper are summarized as follows:

- We present proofs of the correctness of the context-moving transformation for TRSs with respect to both eager evaluation semantics and initial algebra semantics. Moreover, we provide an example to illustrate the usefulness of our result in comparison to [4] (i.e., a transformation for a TRS where the initial algebra semantics differs from the eager evaluation semantics).
- We report on an implementation and experiments of the context-moving and context-splitting transformations for TRSs including non-orthogonal cases. This is novel since [4] does not report on any implementation or experiments.
- Proving the correctness with respect to eager evaluation semantics has not been treated in [8], and our proof of it differs from the one of [4]; we simply use induction on the length of the evaluation while the proof of [4] depends on induction on an unusual ordering (denoted $\succ_f$ in [4]).
- In our proof of the correctness with respect to initial algebra semantics, we do not assume the uniqueness of normal forms nor orthogonality in output TRSs, in contrast to the proofs of the correctness in [4] and [8]. In the proof for the context-splitting transformation, we introduce a new translation $()^\bullet$ between the terms of input and output TRSs besides the translation $()^\circ$ which is the same as one used in [4].

The rest of the paper is organized as follows. Section 2 contains preliminaries. We formulate the context-moving transformation for TRSs and study its correctness in Section 3. We briefly discuss the context-splitting transformation for TRSs in Section 4. We report on an implementation and experiments in Section 5. Section 6 concludes with suggestions for further work.

To save space we omit some of the details in proofs, but a long version of the paper is available at `http://www.nue.riec.tohoku.ac.jp/user/kentaro/`.

2 Preliminaries

In this section, we fix notations and notions used in the paper.

The set of *terms* over *function symbols* $\mathcal{F}$ and *variables* $\mathcal{V}$ is denoted by $\mathcal{T}(\mathcal{F},\mathcal{V})$. The set of variables (function symbols) occurring in a term t is denoted by $\mathcal{V}(t)$ (resp. $\mathcal{F}(t)$). We abbreviate a sequence of terms $t_1,t_2,\ldots,t_n$ as $\bar{t}$; we define $\mathcal{V}(\bar{t}) = \bigcup_{i=1}^{n}\mathcal{V}(t_i)$ and $\mathcal{F}(\bar{t}) = \bigcup_{i=1}^{n}\mathcal{F}(t_i)$. A term t is *ground* if $\mathcal{V}(t)=\emptyset$; the set of ground terms is denoted by $\mathcal{T}(\mathcal{F})$. The *root symbol* of a term t is denoted by $root(t)$. A *context* is a term containing precisely one occurrence of each special constant $\Box_1,\ldots,\Box_n$ (*holes*) for some n. A context C is denoted by $C[\,]$ if $n=1$. If C is a context with n holes then the term obtained by replacing each $\Box_i$ $(1\le i\le n)$ in C with t_i is denoted by $C[t_1,t_2,\ldots,t_n]$. A *substitution* is a function $\theta:\mathcal{V}\to\mathcal{T}(\mathcal{F},\mathcal{V})$ (we omit the usual condition of substitutions that they have a finite domain to ease the notation). A substitution θ is *ground* if $\theta:\mathcal{V}\to\mathcal{T}(\mathcal{F})$; throughout the paper, θ_g,θ'_g, etc. denote ground substitutions.

A *rewrite rule* $l\to r$ satisfies $l\notin\mathcal{V}$ and $\mathcal{V}(l)\supseteq\mathcal{V}(r)$. We assume that variables in rewrite rules are renamed when necessary. A *term rewriting system* (*TRS*, for short) is a finite set of rewrite rules. We call $l\to r$ an *R-rule* if $l\to r\in R$. The set of *defined function symbols* of a TRS R is given by $\mathcal{D}=\{root(l)\mid l\to r\in R\}$ and the set of *constructor symbols* is $\mathcal{C}=\mathcal{F}\setminus\mathcal{D}$. Terms in $\mathcal{T}(\mathcal{C},\mathcal{V})$ are *constructor terms*; terms in $\mathcal{T}(\mathcal{C})$ are *ground constructor terms*. A TRS R is a *constructor TRS* if for any rewrite rule $f(l_1,\ldots,l_n)\to r\in R$, each l_i $(1\le i\le n)$ is a constructor term. A *ground constructor substitution* is a substitution $\theta:\mathcal{V}\to\mathcal{T}(\mathcal{C})$; throughout the paper, θ_{gc},θ'_{gc}, etc. denote ground constructor substitutions.

In this paper, we work with unsorted TRSs for simplicity, but we elaborate lemmas and definitions so that they can be easily adapted to those in the setting of (monomorphic) many-sorted TRSs.

3 Context-moving transformation for TRSs

In this section, we formulate the context-moving transformation for TRSs and prove the correctness of the transformation for some classes of TRSs. In the context-moving transformation for functional programs in [4], the context occurring around the accumulator variable is moved outside of the recursive calls in each rule. The context-moving transformation *for TRSs* follows the same idea.

Definition 1 (context-moving transformation for TRSs). A *context-moving transformation from TRS R to TRS R'* is given as:

$$
\begin{array}{lll}
R = R_A\cup R_B\cup R_C \text{ where } & R_A = \{f(\bar{l}_i,z)\to f(\bar{r}_i,C_i[z]) \mid 1\le i\le m\} \\
& R_B = \{f(\bar{l}_j,z)\to C_j[z] \mid m+1\le j\le n\} \\
& R_C = \{l_k\to r_k \mid n+1\le k\le p\} \\
R' = R'_A\cup R_B\cup R_C \text{ where } & R'_A = \{f(\bar{l}_i,z)\to C_i[f(\bar{r}_i,z)] \mid 1\le i\le m\}
\end{array}
$$

Here, R_A, R'_A consist of m recursive f-rules, R_B consists of $(n-m)$ non-recursive f-rules, and R_C consists of $(p-n)$ other (non f-)rules. The function symbol f is the *target* of the transformation, the contexts $C_1[\ ], \ldots, C_n[\ ]$ are the *moving contexts*, and the variable z is the *accumulator*. Furthermore, *it is required that the target f and the accumulator z do not appear anywhere else except the places explicitly indicated*, i.e., (i) $f \notin (\bigcup_{i=1}^{m} \mathcal{F}(\bar{l}_i, \bar{r}_i, C_i)) \cup (\bigcup_{j=m+1}^{n} \mathcal{F}(\bar{l}_j, C_j)) \cup (\bigcup_{k=n+1}^{p} \mathcal{F}(l_k, r_k))$ and (ii) $z \notin (\bigcup_{i=1}^{m} \mathcal{V}(\bar{l}_i, \bar{r}_i, C_i)) \cup (\bigcup_{j=m+1}^{n} \mathcal{V}(\bar{l}_j, C_j))$.

Henceforth, we will focus on a context-moving transformation from R to R', and unless otherwise stated, $f, z, R_A, C_i, \bar{l}_i, \ldots$ are supposed to be those specified in the definition above.

Example 3 (context-moving transformation). Let R be the following TRS for multiplication.

$$R = \left\{ \begin{array}{ll} (a)\ \mathit{Mult}(S(x), y, z) \to \mathit{Mult}(x, y, \mathit{Add}(y, z)), & (b)\ \mathit{Mult}(0, y, z) \to z \\ (c)\ \mathit{Add}(S(x), y) \to S(\mathit{Add}(x, y)), & (d)\ \mathit{Add}(0, y) \to y \end{array} \right\}$$

We apply the context-moving transformation with *Mult* as the target and z as the accumulator. The rewrite rules of R are partitioned into $R_A = \{(a)\}$, $R_B = \{(b)\}$ and $R_c = \{(c), (d)\}$, and there are two moving contexts, namely $C_1 = \mathit{Add}(y, \Box)$ and $C_2 = \Box$. Thus, by definition, we obtain

$$R'_A = \left\{\ \mathit{Mult}(S(x), y, z) \to \mathit{Add}(y, \mathit{Mult}(x, y, z))\ \right\}$$

Therefore, the following TRS R' is obtained.

$$R' = \left\{ \begin{array}{ll} \mathit{Mult}(S(x), y, z) \to \mathit{Add}(y, \mathit{Mult}(x, y, z)), & \mathit{Mult}(0, y, z) \to z \\ \mathit{Add}(S(x), y) \to S(\mathit{Add}(x, y)), & \mathit{Add}(0, y) \to y \end{array} \right\}$$

The rest of this section is devoted to the discussion on the correctness of the context-moving transformation.

3.1 Correctness of the context-moving transformation with respect to eager evaluation semantics

First we discuss the correctness with respect to eager evaluation semantics as considered in [4]. We assume in this subsection that R is a constructor TRS.

An *eager rewrite relation* $\xrightarrow{e}_R$ is a binary relation on $\mathcal{T}(\mathcal{F})$ given by $s \xrightarrow{e}_R t$ iff $s = C[l\theta_{gc}]$ and $t = C[r\theta_{gc}]$ for some $l \to r \in R$, a context $C[\ ]$ and a ground constructor substitution θ_{gc}. Further, we assume some specific rewrite strategy (e.g. leftmost(-innermost) with rule priority) so that each rewrite step is deterministic. A rewrite step by the deterministic strategy (the *eager evaluation strategy*) is denoted by $s \xrightarrow{\mathrm{ev}}_R t$. The reflexive transitive closure of $\xrightarrow{\mathrm{ev}}_R$ is denoted by $\xrightarrow{\mathrm{ev}*}_R$. A ground term t is said to be *defined in R* if there exists $v \in \mathcal{T}(\mathcal{C})$ such that $t \xrightarrow{\mathrm{ev}*}_R v$; in that case, $|t|^{\mathrm{ev}}_R$ denotes the length of the reduction sequence from t to v. We use $s \overset{\mathrm{ev}}{=}_R t$ to mean that for any $v \in \mathcal{T}(\mathcal{C})$, $s \xrightarrow{\mathrm{ev}*}_R v$ if and only if $t \xrightarrow{\mathrm{ev}*}_R v$. Note that $\overset{\mathrm{ev}}{=}_R$ is an equivalence relation and if $s \xrightarrow{\mathrm{ev}*}_R t$ then $s \overset{\mathrm{ev}}{=}_R t$.

The following are basic properties of an eager evaluation strategy, which are freely used in the rest of this subsection.

Lemma 1. *1. If $s \overset{\mathrm{ev}}{\equiv}_R t$ then $C[s] \overset{\mathrm{ev}}{\equiv}_R C[t]$.*
2. If $C[s]$ is defined in R and $s \overset{\mathrm{ev}}{\to}_R t$ then $|C[s]|^{\mathrm{ev}}_R = |C[t]|^{\mathrm{ev}}_R + 1$.
3. If $C[t]$ is defined in R then so is t, and moreover $|C[t]|^{\mathrm{ev}}_R \geq |t|^{\mathrm{ev}}_R$.

Proof. By induction on $C[\,]$. □

Lemma 2. *For any $l \to r \in R$ and ground substitution θ_g such that $\theta_g(x)$ is defined in R for any $x \in \mathcal{V}(l)$, $l\theta_g \overset{\mathrm{ev}}{\equiv}_R r\theta_g$.*

Proof. If $l\theta_g$ or $r\theta_g$ is defined in R, then $l\theta_g \overset{\mathrm{ev}}{\equiv}_R l\theta_{gc} \overset{\mathrm{ev}}{\equiv}_R r\theta_{gc} \overset{\mathrm{ev}}{\equiv}_R r\theta_g$ for some θ_{gc}. □

We require a property concerning the moving contexts $C_1[\,], \ldots, C_n[\,]$ to guarantee the correctness of the context-moving transformation. This property is given in a similar way to [4] and is formulated as below.

Definition 2 (commutativity law of moving contexts). Let $C_1[\,], \ldots, C_n[\,]$ be the moving contexts of an instance of the context-moving transformation. The *commutativity law of moving contexts* refers to the following condition:

$$\forall i(1 \leq i \leq m).\forall j(1 \leq j \leq n).\forall \theta_{gc}.C_i[C_j[z]]\theta_{gc} \overset{\mathrm{ev}}{\equiv}_R C_j[C_i[z]]\theta_{gc} \qquad (\mathsf{CCOM}^{\mathrm{ev}})$$

Here, we assume that each variable in moving contexts $C_i[\,], C_j[\,]$ is renamed so that their variables do not overlap. By Lemma 1, it is seen that the condition $(\mathsf{CCOM}^{\mathrm{ev}})$ is equivalent to the one with θ_g instead of θ_{gc}.

Example 4 (commutativity law of moving contexts). The moving contexts of the transformation in Example 3 are $C_1 = Add(y, \Box)$ and $C_2 = \Box$ (with $m = 1$ and $n = 2$). As C_2 is a trivial context, the commutativity law of moving contexts is $\forall \theta_{gc}.Add(x, Add(y, z))\theta_{gc} \overset{\mathrm{ev}}{\equiv}_R Add(y, Add(x, z))\theta_{gc}$.

Definition 3 ($R \overset{\mathrm{ev}\,f}{\Rightarrow}_{\mathrm{cm}} R'$). We write $R \overset{\mathrm{ev}\,f}{\Rightarrow}_{\mathrm{cm}} R'$ if R' is obtained from a constructor TRS R by the context-moving transformation such that f is the target and the condition $(\mathsf{CCOM}^{\mathrm{ev}})$ holds.

The commutativity law of moving contexts is essential for guaranteeing the simulation of rewrite sequences from ground terms to ground constructor terms on R by R' and vice versa. The key property to the simulation is the following context-moving lemma.

Lemma 3 (context-moving lemma). *Suppose $R \overset{\mathrm{ev}\,f}{\Rightarrow}_{\mathrm{cm}} R'$. Let $1 \leq i \leq m$, and let θ_{gc} be a ground constructor substitution and $\bar{t}, u$ be ground terms.*

1. If $C_i\theta_{gc}[f(\bar{t}, u)] \overset{\mathrm{ev}}{\to}^*_R v \in \mathcal{T}(\mathcal{C})$ then $f(\bar{t}, C_i\theta_{gc}[u]) \overset{\mathrm{ev}}{\to}^*_R v$.
2. If $f(\bar{t}, C_i\theta_{gc}[u]) \overset{\mathrm{ev}}{\to}^*_{R'} v \in \mathcal{T}(\mathcal{C})$ then $C_i\theta_{gc}[f(\bar{t}, u)] \overset{\mathrm{ev}}{\to}^*_{R'} v$.

Proof. 1. If $C_i\theta_{gc}[f(\bar{t},u)] \xrightarrow{\mathrm{ev}*}_R v \in \mathcal{T}(\mathcal{C})$ then $f(\bar{t},u)$ is defined in R. The claim is proved by induction on $|f(\bar{t},u)|^{\mathrm{ev}}_R$.
2. By induction on $|f(\bar{t},C_i\theta_{gc}[u])|^{\mathrm{ev}}_{R'}$. □

We are now ready to prove the correctness of the context-moving transformation with respect to eager evaluation semantics.

Theorem 1 (correctness of context-moving transformation). *Let R be a constructor TRS. Suppose $R \overset{\mathrm{ev}\,f}{\Rightarrow}_{\mathrm{cm}} R'$. For any ground term s and ground constructor term v, $s \xrightarrow{\mathrm{ev}*}_R v$ if and only if $s \xrightarrow{\mathrm{ev}*}_{R'} v$.*

Proof. By induction on the length of the evaluation, using Lemma 3. □

Remark 1. The proof of the "only if"-part of Theorem 1 given in [4] is based on the converse of Lemma 3.1. For those proofs, induction on an unusual ordering $\succ_f$ is used. In contrast, our proof is based on Lemma 3.2, and it suffices to use induction on the length of the evaluation.

3.2 Correctness of the context-moving transformation with respect to initial algebra semantics

The correctness theorem in the previous subsection depends on the condition ($\mathsf{CCOM}^{\mathrm{ev}}$), which involves a notion of evaluation and does not necessarily correspond to equality in the initial algebra. In this subsection, we show the correctness of the context-moving transformation based on a condition that precisely corresponds to equality in the initial algebra.

First we introduce some standard definitions in term rewriting. A *rewrite relation* $\to_R$ is a binary relation on $\mathcal{T}(\mathcal{F},\mathcal{V})$ given by $s \to_R t$ iff $s = C[l\theta]$ and $t = C[r\theta]$ for some $l \to r \in R$, a context $C[\,]$ and a substitution θ. The reflexive transitive closure of $\to_R$ is denoted by $\overset{*}{\to}_R$. If a unique normal form of t exists, then the normal form of t is denoted by $t{\downarrow_R}$. For each substitution θ, the substitution $\theta{\downarrow_R}$ is defined by $\theta{\downarrow_R}(x) = (\theta(x)){\downarrow_R}$, provided that $(\theta(x)){\downarrow_R}$ is defined for any $x \in \mathcal{V}$. We use $\theta_{g\setminus f}$ to denote a ground substitution such that f does not appear in its range. A TRS R is *sufficiently complete* if $\forall s \in \mathcal{T}(\mathcal{F}).\, \exists v \in \mathcal{T}(\mathcal{C}).\, s \overset{*}{\to}_R v$ holds [6]; R is *ground confluent* if $\overset{*}{\leftarrow}_R \circ \overset{*}{\to}_R \subseteq \overset{*}{\to}_R \circ \overset{*}{\leftarrow}_R$ on $\mathcal{T}(\mathcal{F})$. We assume in this subsection that R is a sufficiently complete and ground confluent TRS.[1]

Now we introduce a property on the moving contexts $C_1[\,], \ldots, C_n[\,]$ to guarantee the correctness with respect to semantics considered in this subsection.

Definition 4 (commutativity law of moving contexts). Let $C_1[\,], \ldots, C_n[\,]$ be the moving contexts of an instance of the context-moving transformation. The *commutativity law of moving contexts* refers to the following condition:

$$\forall i(1 \le i \le m).\forall j(1 \le j \le n).\forall \theta_g.C_i[C_j[z]]\theta_g{\downarrow_R} = C_j[C_i[z]]\theta_g{\downarrow_R} \qquad (\mathsf{CCOM})$$

[1] In the case of many-sorted TRSs, we assume sufficient completeness only for the sort of return values of the target f of the context-moving transformation, meaning that any ground term of that sort can be rewritten to a constructor term. Cf. Example 6.

Here, we assume that each variable in moving contexts $C_i[\,], C_j[\,]$ is renamed so that their variables do not overlap.

In contrast to the condition ($\mathsf{CCOM}^{\mathrm{ev}}$) in Definition 2, the above condition (CCOM) precisely corresponds to equations that are valid in the initial algebra of the input TRS R, i.e. inductive theorems of R, and so may be checked by an inductive theorem prover. (For an actual implementation, see Section 5.)

Definition 5 ($R \Rightarrow^{f}_{\mathrm{cm}} R'$). We write $R \Rightarrow^{f}_{\mathrm{cm}} R'$ if R' is obtained from a sufficiently complete and ground confluent TRS R by the context-moving transformation such that f is the target and the condition (CCOM) holds.

The condition (CCOM) is essential for guaranteeing the simulation of rewrite sequences from ground terms to ground constructor terms on R by R'. We first show the simulation of rewrite sequences of the form $f(\bar{x}, z)\theta_{g\setminus f} \xrightarrow{*}_R v$ (Lemma 4) and then generalize it to an arbitrary case (Lemma 5).

Lemma 4. *Suppose $R \Rightarrow^{f}_{\mathrm{cm}} R'$. For any ground substitution $\theta_{g\setminus f}$ and ground constructor term v, if $f(\bar{x}, z)\theta_{g\setminus f} \xrightarrow{*}_R v$ then $f(\bar{x}, z)\theta_{g\setminus f} \xrightarrow{*}_{R'} v$.*

Proof. Suppose $f(\bar{x}, z)\theta_{g\setminus f} \xrightarrow{*}_R v$. By the form of the rewrite rules in R, we know that any rewrite sequence α of R from $f(\bar{x}, z)\theta_{g\setminus f}$ to v has the following form:

$$\begin{aligned}\alpha : f(\bar{x}, z)\theta_{g\setminus f} \;=\;& f(\bar{l}_{i_1}\theta_1, u_1) \to_{R_A} f(\bar{r}_{i_1}\theta_1, C_{i_1}\theta_1[u_1]) \\ &\xrightarrow{*}_{R_C} f(\bar{l}_{i_2}\theta_2, u_2) \to_{R_A} f(\bar{r}_{i_2}\theta_2, C_{i_2}\theta_1[u_2]) \\ &\qquad\vdots \\ &\xrightarrow{*}_{R_C} f(\bar{l}_{i_n}\theta_n, u_n) \to_{R_B} C_{i_n}\theta_n[u_n] \xrightarrow{*}_{R_C} v\end{aligned}$$

Here $\theta_1, \ldots, \theta_n$ are ground substitutions such that f does not appear in their ranges. Note that rewrite rules of R_A, R_B applicable to any term $f(\bar{t}, u)$ (at root position) are completely specified by $\bar{t}$ regardless of u. Hence, in the rewrite sequence α, the applications of R_A, R_B-rules are not affected even if one postpones the applications of R_C-rules to u_i's. Thus, one can obtain the next rewrite sequence β from α, by distinguishing the applications of R_C-rules to the last argument of f and those to the rest, and postponing the former:

$$\begin{aligned}\beta : f(\bar{x}, z)\theta_{g\setminus f} \;=\;& f(\bar{l}_{i_1}\theta_1, u_1) \to_{R_A} f(\bar{r}_{i_1}\theta_1, C_{i_1}\theta_1[u_1]) \\ &\xrightarrow{*}_{R_C} f(\bar{l}_{i_2}\theta_2, C_{i_1}\theta_1[u_1]) \to_{R_A} f(\bar{r}_{i_2}\theta_2, C_{i_2}\theta_2[C_{i_1}\theta_1[u_1]]) \\ &\qquad\vdots \\ &\xrightarrow{*}_{R_C} f(\bar{l}_{i_n}\theta_n, C_{i_{n-1}}\theta_{n-1}[\cdots C_{i_1}\theta_1[u_1]\cdots]) \\ &\qquad\to_{R_B} C_{i_n}\theta_n[C_{i_{n-1}}\theta_{n-1}[\cdots C_{i_1}\theta_1[u_1]\cdots]] \xrightarrow{*}_{R_C} v\end{aligned}$$

Next we construct a rewrite sequence γ of R' from β (of R). It is easy to observe in the definition of context-moving transformation that for any i and θ, $f(\bar{l}_i, z)\theta \to_{R_A} f(\bar{r}_i, C_i[z])\theta$ implies $f(\bar{l}_i, z)\theta \to_{R'_A} C_i[f(\bar{r}_i, z)]\theta$. Thus, by moving out the contexts $C_{i_j}\theta_j[\,]$ in each R_A-step, we obtain the corresponding R'_A-step.

Then the next rewrite sequence γ is obtained from β.

$$\begin{aligned}\gamma : f(\bar{x}, z)\theta_{g\setminus f} &= f(\bar{l}_{i_1}\theta_1, u_1) \to_{R'_A} C_{i_1}\theta_1[f(\bar{r}_{i_1}\theta_1, u_1)] \\ &\xrightarrow{*}_{R_C} C_{i_1}\theta_1[f(\bar{l}_{i_2}\theta_2, u_1)] \to_{R'_A} C_{i_1}\theta_1[C_{i_2}\theta_2[f(\bar{r}_{i_2}\theta_2, u_1)]] \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vdots \\ &\xrightarrow{*}_{R_C} C_{i_1}\theta_1[\cdots C_{i_{n-1}}\theta_{n-1}[f(\bar{l}_{i_n}\theta_n, u_1)]\cdots] \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \to_{R_B} C_{i_1}\theta_1[\cdots C_{i_{n-1}}\theta_{n-1}[C_{i_n}\theta_n[u_1]]\cdots]\end{aligned}$$

Since R is ground confluent, so is R_C. Thus, by the condition (CCOM) and $f \notin \mathcal{F}(C_{i_1}\theta_1[\cdots C_{i_{n-1}}\theta_{n-1}[C_{i_n}\theta_n[u_1]]\cdots])$, it follows

$$\begin{aligned}v &= C_{i_n}\theta_n[C_{i_{n-1}}\theta_{n-1}[\cdots C_{i_1}\theta_1[u_1]\cdots]]\downarrow_{R_C} \\ &= C_{i_1}\theta_1[\cdots C_{i_{n-1}}\theta_{n-1}[C_{i_n}\theta_n[u_1]]\cdots]\downarrow_{R_C}\end{aligned}$$

Hence, we obtain $f(\bar{x}, z)\theta_{g\setminus f} \xrightarrow{*}_{R'} C_{i_1}\theta_1[\cdots C_{i_{n-1}}\theta_{n-1}[C_{i_n}\theta_n[u_1]]\cdots] \xrightarrow{*}_{R_C} v$. □

Lemma 5. *Suppose $R \Rightarrow^f_{\mathrm{cm}} R'$. For any ground term s and ground constructor term v, if $s \xrightarrow{*}_R v$ then $s \xrightarrow{*}_{R'} v$.*

Proof. By induction on the number of occurrences of f in s, using Lemma 4. □

In contrast to the proof in the previous subsection, a key ingredient of the proof of the correctness here is preservation of two properties of R: sufficient completeness and ground confluence. The former is a direct consequence of Lemma 5.

Lemma 6. *Suppose $R \Rightarrow^f_{\mathrm{cm}} R'$. Then R' is sufficiently complete.*

Proof. It follows by Lemma 5 from the sufficient completeness of R. □

To show preservation of ground confluence, we need the simulation of rewrite sequences from ground terms to ground constructor terms on R' by R, that is, the converse of Lemma 5. To this end, we first prove the following lemma, where we use again the forms of rewrite rules in R and R' and the condition (CCOM).

Lemma 7. *Suppose $R \Rightarrow^f_{\mathrm{cm}} R'$. For any ground terms s, s' and ground constructor term v, if $s \xrightarrow{*}_R v$ and $s \to_{R'} s'$ then $s' \xrightarrow{*}_R v$.*

Proof. If $s \to_{R_B \cup R_C} s'$, then $s \to_R s'$ by $R_B \cup R_C \subseteq R$, and hence the claim follows immediately by the ground confluence of R. It remains to prove the case $s \to_{R'_A} s'$. Then one has $s = C[f(\bar{l}_{i_1}\theta_1, u)]$ and $s' = C[C_{i_1}\theta_1[f(\bar{r}_{i_1}\theta_1, u)]]$, and thus, $s = C[f(\bar{l}_{i_1}\theta_1, u)] \to_{R_A} C[f(\bar{r}_{i_1}\theta_1, C_{i_1}\theta_1[u])]$. Furthermore, since $s \xrightarrow{*}_R v$, it follows from the ground confluence of R that $C[f(\bar{r}_{i_1}\theta_1, C_{i_1}\theta_1[u])] \xrightarrow{*}_R v$. Thus, $s = C[f(\bar{l}_{i_1}\theta_1, u)] \to_{R_A} C[f(\bar{r}_{i_1}\theta_1, C_{i_1}\theta_1[u])] \xrightarrow{*}_R v$. Now, as in the proof of Lemma 4, this rewrite sequence looks like:

$$\begin{aligned}\alpha : s &= C[f(\bar{l}_{i_1}\theta_1, u)] \to_{R_A} C[f(\bar{r}_{i_1}\theta_1, C_{i_1}\theta_1[u])] \\ &\xrightarrow{*}_{R_C} C[f(\bar{l}_{i_2}\theta_2, C_{i_1}\theta_1[u])] \to_{R_A} C[f(\bar{r}_{i_2}\theta_2, C_{i_2}\theta_2[C_{i_1}\theta_1[u]])] \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vdots \\ &\xrightarrow{*}_{R_C} C[f(\bar{l}_{i_n}\theta_n, C_{i_{n-1}}\theta_{n-1}[\cdots C_{i_1}\theta_1[u]\cdots])] \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \to_{R_B} C[C_{i_n}\theta_n[C_{i_{n-1}}\theta_{n-1}[\cdots C_{i_1}\theta_1[u]\cdots]]] \xrightarrow{*}_R v\end{aligned}$$

Consider the next rewrite sequence β obtained from α by replacing the first R_A-step with an R'_A-step:

$$\begin{aligned}
\beta : s \;=\;& C[f(\bar{l}_{i_1}\theta_1, u)] \to_{R'_A} C[C_{i_1}\theta_1[f(\bar{r}_{i_1}\theta_1, u)]] \\
&\xrightarrow{*}_{R_C} C[C_{i_1}\theta_1[f(\bar{l}_{i_2}\theta_2, u)]] \to_{R_A} C[C_{i_1}\theta_1[f(\bar{r}_{i_2}\theta_2, C_{i_2}\theta_2[u])]] \\
&\xrightarrow{*}_{R_C} C[C_{i_1}\theta_1[f(\bar{l}_{i_3}\theta_3, C_{i_2}\theta_2[u])]] \to_{R_A} C[C_{i_1}\theta_1[f(\bar{r}_{i_3}\theta_3, C_{i_3}\theta_3[C_{i_2}\theta_2[u]])]] \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vdots \\
&\xrightarrow{*}_{R_C} C[C_{i_1}\theta_1[f(\bar{l}_{i_n}\theta_n, C_{i_{n-1}}\theta_{n-1}[\cdots C_{i_2}\theta_2[u]\cdots])]] \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \to_{R_B} C[C_{i_1}\theta_1[C_{i_n}\theta_n[C_{i_{n-1}}\theta_{n-1}[\cdots C_{i_2}\theta_2[u]\cdots]]]]
\end{aligned}$$

Then, by the condition (CCOM) and ground confluence of R, we have

$$\begin{aligned}
v &= C[C_{i_n}\theta_n[C_{i_{n-1}}\theta_{n-1}[\cdots C_{i_1}\theta_1[u]\cdots]]]{\downarrow_R} \\
&= C[C_{i_1}\theta_1[C_{i_n}\theta_n[C_{i_{n-1}}\theta_{n-1}[\cdots C_{i_2}\theta_2[u]\cdots]]]]{\downarrow_R}
\end{aligned}$$

Since $s' = C[C_{i_1}\theta_1[f(\bar{r}_{i_1}\theta_1, u)]] \xrightarrow{*}_R C[C_{i_1}\theta_1[C_{i_n}\theta_n[C_{i_{n-1}}\theta_{n-1}[\cdots C_{i_2}\theta_2[u]\cdots]]]]$ (in β), we conclude $s' \xrightarrow{*}_R v$. □

The rewrite step $s \to_{R'} s'$ in the above lemma can be generalized to $s \xrightarrow{*}_{R'} s'$.

Lemma 8. *Suppose $R \Rightarrow^f_{\mathrm{cm}} R'$. For any ground terms s, s' and ground constructor term v, if $s \xrightarrow{*}_R v$ and $s \xrightarrow{*}_{R'} s'$ then $s' \xrightarrow{*}_R v$.*

Now we can prove the converse of Lemma 5.

Lemma 9. *Suppose $R \Rightarrow^f_{\mathrm{cm}} R'$. For any ground term s and ground constructor term v, if $s \xrightarrow{*}_{R'} v$ then $s \xrightarrow{*}_R v$.*

Proof. By sufficient completeness of R, there exists a ground constructor term v' such that $s \xrightarrow{*}_R v'$. By Lemma 8, we have $v \xrightarrow{*}_R v'$, and thus $v = v'$ as v is a constructor term. Hence, $s \xrightarrow{*}_R v$. □

Now we arrive at the preservation of ground confluence.

Lemma 10. *Suppose $R \Rightarrow^f_{\mathrm{cm}} R'$. Then R' is ground confluent.*

Proof. Let t be a ground term and suppose that $t \xrightarrow{*}_{R'} t_1$ and $t \xrightarrow{*}_{R'} t_2$. Since R' is sufficiently complete by Lemma 6, there exist ground constructor terms v_1, v_2 such that $t_1 \xrightarrow{*}_{R'} v_1$ and $t_2 \xrightarrow{*}_{R'} v_2$. By Lemma 9, we have $t \xrightarrow{*}_R v_1$ and $t \xrightarrow{*}_R v_2$. Then by ground confluence of R, we obtain $v_1 = v_2$. Hence R' is ground confluent. □

We are now ready to show the main theorem of this subsection, which implies that the context-moving transformation preserves equality in the initial algebra and the terms in each equivalence class have the same normal form by R and R'.

Theorem 2 (correctness of context-moving transformation). *Let R be a sufficiently complete and ground confluent TRS. Suppose $R \Rightarrow^f_{\mathrm{cm}} R'$. Then for any ground term s, $s{\downarrow_R} = s{\downarrow_{R'}}$.*

Proof. By sufficient completeness and ground confluence of R and R', $s{\downarrow_R}$ and $s{\downarrow_{R'}}$ are unique constructor ground terms. By Lemma 5, we have $s \xrightarrow{*}_{R'} s{\downarrow_R}$. Thus, $s{\downarrow_R} = s{\downarrow_{R'}}$. □

Example 5 (context-moving transformation for non-orthogonal system). Let R be the following non-orthogonal TRS for a list calculation.

$$R = \left\{ \begin{array}{ll} (a) & \mathit{Minlist}(\mathit{Cons}(x, xs), z) \to \mathit{Minlist}(xs, \mathit{Min}(x, z)) \\ (b) & \mathit{Minlist}(\mathit{Nil}, z) \to z \\ (c) & \mathit{Min}(S(x), S(y)) \to S(\mathit{Min}(x, y)) \\ (d) & \mathit{Min}(0, y) \to 0, \quad (e) \quad \mathit{Min}(x, 0) \to 0 \end{array} \right\}$$

where we assume that it is many-sorted with sorts *Nat* and *NatList* in an appropriate way. We apply the context-moving transformation with *Minlist* as the target and z as the accumulator. We have $R_C = \{(c), (d), (e)\}$ and there are two moving contexts, namely $C_1 = \mathit{Min}(x, \Box)$ and $C_2 = \Box$. Then we have

$$\forall \theta_g . \mathit{Min}(x, \mathit{Min}(y, z))\theta_g{\downarrow_R} = \mathit{Min}(y, \mathit{Min}(x, z))\theta_g{\downarrow_R}$$

and thus, $R \Rightarrow_{\mathrm{cm}}^{\mathit{Minlist}} R'$, where

$$R' = \{\mathit{Minlist}(\mathit{Cons}(x, xs), z) \to \mathit{Min}(x, \mathit{Minlist}(xs, z))\} \cup \{(b)\text{–}(e)\}$$

Example 6. In this example, we use a many-sorted TRS R with sorts *Nat* and *NatStream*, where ":" of sort $\mathit{Nat} \times \mathit{NatStream} \to \mathit{NatStream}$ is the only constructor symbol for terms of sort *NatStream*.

$$R = \left\{ \begin{array}{ll} (a) & \mathit{Sum}(S(x), \alpha, z) \to \mathit{Sum}(x, \mathit{Tl}(\alpha), \mathit{Add}(\mathit{Hd}(\alpha), z)) \\ (b) & \mathit{Sum}(0, \alpha, z) \to z \\ (c) & \mathit{Hd}(x : \alpha) \to x, \quad (d) \quad \mathit{Tl}(x : \alpha) \to \alpha \\ (e) & \mathit{Inc} \to 0 : \mathit{Succ}(\mathit{Inc}), \quad (f) \quad \mathit{Succ}(x : \alpha) \to S(x) : \mathit{Succ}(\alpha) \\ (g) & \mathit{Add}(S(x), y) \to S(\mathit{Add}(x, y)), \quad (h) \quad \mathit{Add}(0, y) \to y \end{array} \right\}$$

Here we have sufficient completeness for sort *Nat*, which is the sort of return values of the target *Sum*. Then, all the arguments for the correctness of the context-moving transformation follows for terms of sort *Nat*.[2] We have $R_C = \{(c)\text{–}(h)\}$ and there are two moving contexts, namely $C_1 = \mathit{Add}(\mathit{Hd}(\alpha), \Box)$ and $C_2 = \Box$. Then we have

$$\forall \theta_g . \mathit{Add}(\mathit{Hd}(\alpha), \mathit{Add}(\mathit{Hd}(\beta), z))\theta_g{\downarrow_R} = \mathit{Add}(\mathit{Hd}(\beta), \mathit{Add}(\mathit{Hd}(\alpha), z))\theta_g{\downarrow_R}$$

Thus, we obtain $R \Rightarrow_{\mathrm{cm}}^{\mathit{Sum}} R'$, where

$$R' = \{\mathit{Sum}(S(x), \alpha, z) \to \mathit{Add}(\mathit{Hd}(\alpha), \mathit{Sum}(x, \mathit{Tl}(\alpha), z))\} \cup \{(b)\text{–}(h)\}$$

Note here that, for terms of sort *Nat*, normal forms may not be reached by the eager evaluation strategy because of the rule for *Inc*.

[2] For terms of sort *NatStream*, we do not seek the correctness of the context-moving transformation in the style of Theorem 2.

4 Context-splitting transformation for TRSs

In this section, we formulate the context-splitting transformation for TRSs and prove the correctness of the transformation. In the context-splitting transformation for functional programs in [4], the context occurring around the accumulator variable is required to be split into a "common" part and an "own" part, where the "common" part needs to be common to all f-rules. Then, in each rule, the context is moved outside of the recursive calls, moving its own part and removing the accumulator. Furthermore, the target f is replaced with a new function symbol f', obtained by removing the accumulator argument of f. The context-splitting transformation *for TRSs* follows the same idea.

Definition 6 (context-splitting transformation for TRSs). The *context-splitting transformation from TRS R to TRS R'* is given as:

$$R = R_A \cup R_B \cup R_C \text{ where } \begin{array}{l} R_A = \{f(\bar{l}_i, z) \to f(\bar{q}_i, C_i[z]) \mid 1 \le i \le m\} \\ R_B = \{f(\bar{l}_j, z) \to C_j[z] \mid m+1 \le j \le n\} \\ R_C = \{l_k \to r_k \mid n+1 \le k \le p\} \end{array}$$

For each i $(1 \le i \le n)$, it is required that either $C_i[\,] = C[r_i, \Box]$ or $C_i[\,] = \Box$.

$$R' = R'_A \cup R'_B \cup R_C \text{ where } \begin{array}{l} R'_A = \{f'(\bar{l}_i) \to C'_i[f'(\bar{q}_i)] \mid 1 \le i \le m\} \\ R'_B = \{f'(\bar{l}_j) \to r'_j \mid m+1 \le j \le n\} \end{array}$$

Here, for each i $(1 \le i \le m)$ and j $(m+1 \le j \le n)$, the context $C'_i[\,]$ and the term r'_j are given like this:

$$C'_i[\,] = \begin{cases} C[\Box, r_i] & \text{if } C_i[\,] = C[r_i, \Box] \\ \Box & \text{if } C_i[\,] = \Box \end{cases} \qquad r'_j = \begin{cases} r_j & \text{if } C_j[\,] = C[r_j, \Box] \\ e & \text{if } C_j[\,] = \Box \end{cases}$$

The function symbol f is the *target* of the transformation, the variable z is the *accumulator*, the context C is the *common context*, and the term e is the *unit*. Here, the common context C should be a ground context such that $f \notin \mathcal{F}(C)$ and the unit e should be a ground constructor term. Furthermore, *it is required that the target f and the accumulator z do not appear anywhere else except the places explicitly indicated.*

Example 7 (context-splitting transformation). Let R be the following TRS for list concatenation. Here we assume that it is many-sorted in an appropriate way.

$$R = \left\{ \begin{array}{ll} (a)\ Cat(LCons(x, xs), z) \to Cat(xs, App(z, x)), & (b)\ Cat(LNil, z) \to z \\ (c)\ App(Cons(x, xs), y) \to Cons(x, App(xs, y)), & (d)\ App(Nil, y) \to y \end{array} \right\}$$

We apply the context-splitting transformation with Cat as the target and z as the accumulator. The TRS R is partitioned like this: $R_A = \{(a)\}$, $R_B = \{(b)\}$ and $R_C = \{(c), (d)\}$. We remark that the common context is $C = App(\Box_2, \Box_1)$ and we have $C_1[\,] = App(\Box, x)$ and $C_2[\,] = \Box$. The unit is $e = Nil$. We construct R'_A, R'_B from R_A, R_B as follows:

$$R'_A = \left\{\, Cat'(LCons(x, xs)) \to App(x, Cat'(xs)) \,\right\} \qquad R'_B = \left\{\, Cat'(LNil) \to Nil \,\right\}$$

Thus, we obtain $R' = R'_A \cup R'_B \cup \{(c), (d)\}$.

4.1 Correctness of the context-splitting transformation with respect to initial algebra semantics

In the context-moving transformation, the commutativity laws of moving contexts played an important role. In the context-splitting transformation, we require two conditions instead: "associativity law of common context" and "unit law of common context"; they are defined again following [4] but in the forms that correspond to equality in the initial algebra as (CCOM) in Definition 4.

Definition 7 (associativity law of common context). Let C be the common context of an instance of the context-splitting transformation. The *associativity law of common context* for the transformation refers to the following condition:

$$\forall\theta_g.C[C[x,y],z]\theta_g{\downarrow_R} = C[x,C[y,z]]\theta_g{\downarrow_R} \qquad (\mathsf{CASSOC})$$

Definition 8 (unit law of common context). Let C be the common context and e be the unit of an instance of the context-splitting transformation. The *unit law of common context* for the transformation refers to the following condition:

$$\forall\theta_g.C[x,e]\theta_g{\downarrow_R} = C[e,x]\theta_g{\downarrow_R} = \theta_g(x) \qquad (\mathsf{CUNIT})$$

Definition 9 ($R \Rightarrow^f_{\mathrm{cs}} R'$). We write $R \Rightarrow^f_{\mathrm{cs}} R'$ if R' is obtained from a sufficiently complete and ground confluent TRS R by the context-splitting transformation such that f is the target and the conditions (CASSOC) and (CUNIT) hold.

Definition 10 (translations $()^\circ, ()^\bullet$). Let C be the common context and e be the unit of an instance of the context-splitting transformation. We recursively define the term $t^\circ \in \mathcal{T}(\mathcal{F}',\mathcal{V})$ for each term $t \in \mathcal{T}(\mathcal{F},\mathcal{V})$ and the term $t^\bullet \in \mathcal{T}(\mathcal{F},\mathcal{V})$ for each term $t \in \mathcal{T}(\mathcal{F}',\mathcal{V})$ as follows:

$$t^\circ = \begin{cases} C[f'(\bar{t}^\circ), u^\circ] & \text{if } t = f(\bar{t},u) \\ g(\bar{t}^\circ) & \text{if } t = g(\bar{t}), g \neq f \\ t & \text{if } t \in \mathcal{V} \end{cases} \qquad t^\bullet = \begin{cases} f(\bar{t}^\bullet, e) & \text{if } t = f'(\bar{t}) \\ g(\bar{t}^\bullet) & \text{if } t = g(\bar{t}), g \neq f' \\ t & \text{if } t \in \mathcal{V} \end{cases}$$

Here, for each sequence $\bar{t} = t_1,\ldots,t_n$, we let $\bar{t}^\star = t_1^\star,\ldots,t_n^\star$ for $\star \in \{\circ,\bullet\}$.

We first show two kinds of simulation of rewrite sequences from ground terms to ground constructor terms on R by R'.

Lemma 11. *Suppose $R \Rightarrow^f_{\mathrm{cs}} R'$. For any ground term s and ground constructor term v, (i) if $s \xrightarrow{*}_R v$ then $s^\circ \xrightarrow{*}_{R'} v$ and (ii) if $s^\bullet \xrightarrow{*}_R v$ then $s \xrightarrow{*}_{R'} v$.*

Using Lemma 11, we can show the correctness of the context-splitting transformation. As in the case of the context-moving transformation, a key ingredient of the proof is preservation of sufficient completeness and ground confluence.

Lemma 12. *Suppose $R \Rightarrow^f_{\mathrm{cs}} R'$. Then R' is sufficiently complete.*

Lemma 13. *Suppose $R \Rightarrow^f_{\mathrm{cs}} R'$. Then R' is ground confluent.*

Theorem 3 (correctness of context-splitting transformation). *Let R be a sufficiently complete and ground confluent TRS. Suppose $R \Rightarrow^f_{\mathrm{cs}} R'$. Then for any ground term s, $s{\downarrow_R} = s^\circ{\downarrow_{R'}}$.*

5 Automating the context-moving and context-splitting transformations

In this section, we report on an implementation and experiments of the context-moving and context-splitting transformations for TRSs presented in this paper. A key feature of our implementation is to employ inductive theorem proving to verify the commutative law of moving contexts, etc. to guarantee the correctness of the transformations.

An equation $s \doteq t$ is an *inductive theorem* of a TRS R ($R \models_{\mathrm{ind}} s \doteq t$) if $s\theta_g \overset{*}{\leftrightarrow}_R t\theta_g$ for any ground substitution θ_g. It is known that these equations coincide with the equations that are valid in the initial algebra of R. The next lemma follows immediately from the definition.

Lemma 14. *Let R be a sufficiently complete and ground confluent TRS. Then $R \models_{\mathrm{ind}} s \doteq t$ iff for any ground substitution θ_g, $s\theta_g{\downarrow_R} = t\theta_g{\downarrow_R}$.*

Thus, the commutative law of moving contexts and the associative and unit laws of common context, in the forms of (CCOM) in Definition 4, (CASSOC) in Definition 7 and (CUNIT) in Definition 8, are guaranteed if one succeeds in proving the following conditions (C), (A) and (U), respectively.

$$\begin{array}{ll} \text{(C)} & \forall i(1 \le i \le m)\forall j(1 \le j \le n).R \models_{\mathrm{ind}} C_i[C_j[z]] \doteq C_j[C_i[z]] \\ \text{(A)} & R \models_{\mathrm{ind}} C[C[x,y],z] \doteq C[x,C[y,z]] \\ \text{(U)} & R \models_{\mathrm{ind}} C[x,e] \doteq x,\ R \models_{\mathrm{ind}} C[e,x] \doteq x \end{array}$$

Here, $C_1, \ldots, C_m$ are the moving contexts, C is the common context, and e is the unit of the transformation.

We have implemented a TRS transformation procedure with the context-moving and context-splitting transformations using Standard ML of New Jersey. We employed rewriting induction [7] for proving conditions (C), (A) and (U). Since one generally needs to deal with non-orientable equations for proving the condition (C), we have used rewriting induction for non-orientable equations [1].

We have tested context-moving transformations, context-splitting transformations, and their combinations. Among 21 examples, the context-moving transformations succeeded at 15 examples and the context-splitting transformations succeeded at 10 examples. There are 6 examples which succeeded in both of the transformations. Failure of 3 examples in context-moving transformations and 4 in context-splitting transformations are due to failure of rewriting induction.

All details of the experiments are available on the webpage `http://www.nue.riec.tohoku.ac.jp/tools/experiments/lopstr15/`.

6 Conclusion

We have presented proofs of the correctness of context-moving and context-splitting transformations for TRSs. First we gave a proof of the correctness of the context-moving transformation with respect to eager evaluation semantics as

considered in [4]. Then we gave proofs of the correctness of the context-moving and context-splitting transformations with respect to initial algebra semantics, where the conditions of the transformations precisely correspond to equality in the initial algebra and so can be checked by an inductive theorem prover.

The context-moving transformation for TRSs with eager evaluation as well as the transformations in [4] allows input programs where a term may not be evaluated to a ground constructor term either because it is not terminating under the evaluation strategy or because evaluation gets stuck at a non-constructor term. To deal with such programs in general (i.e., to prove their properties and to check the conditions for the correctness of the transformations), one needs methods for induction proofs with partial functions as studied in [5]. Also, the correctness of the transformations for programs with other evaluation strategies, e.g. lazy evaluation, is to be investigated. These problems and their implementation are left as future work.

Acknowledgements. We are grateful to the anonymous referees for valuable comments. This research was supported by JSPS KAKENHI Grant Numbers 25330004, 25280025 and 15K00003.

References

1. T. Aoto. Designing a rewriting induction prover with an increased capability of non-orientable equations. In *Proc. of 1st SCSS*, volume 08-08 of *RISC Technical Report*, pages 1–15, 2008.
2. T. Aoto. Sound lemma generation for proving inductive validity of equations. In *Proc. of 28th FSTTCS*, volume 2 of *LIPIcs*, pages 13–24. Schloss Dagstuhl, 2008.
3. A. Bouhoula, E. Kounalis, and M. Rusinowitch. Automated mathematical induction. *Journal of Logic and Computation*, 5(5):631–668, 1995.
4. J. Giesl. Context-moving transformations for function verification. In *Proc. of 9th LOPSTR*, volume 1817 of *LNCS*, pages 293–312. Springer-Verlag, 2000.
5. J. Giesl. Induction proofs with partial functions. *Journal of Automated Reasoning*, 26(1):1–49, 2001.
6. D. Kapur, P. Narendran, and H. Zhang. On sufficient-completeness and related properties of term rewriting systems. *Acta Informatica*, 24(4):395–415, 1987.
7. U. S. Reddy. Term rewriting induction. In *Proc. of 10th CADE*, volume 449 of *LNAI*, pages 162–177. Springer-Verlag, 1990.
8. K. Sato, K. Kikuchi, T. Aoto, and Y. Toyama. Automated inductive theorem proving using transformations of term rewriting systems. *JSSST Computer Software*, 32(1):179–193, 2015. In Japanese.
9. S. Shimazu, T. Aoto, and Y. Toyama. Automated lemma generation for rewriting induction with disproof. *JSSST Computer Software*, 26(2):41–55, 2009. In Japanese.
10. S. Stratulat. A general framework to build contextual cover set induction provers. *Journal of Symbolic Computation*, 32:403–445, 2001.
11. P. Urso and E. Kounalis. Sound generalizations in mathematical induction. *Theoretical Computer Science*, 323:443–471, 2004.
12. T. Walsh. A divergence critic for inductive proof. *Journal of Artificial Intelligence Research*, 4:209–235, 1996.