PAPER Special Section on Formal Approach

Program Transformation Templates for Tupling Based on Term Rewriting

Yuki CHIBA[†a)], **Takahito AOTO**[††b)], ***Nonmembers***, ***and*** **Yoshihito TOYAMA**[††c)], ***Member***

SUMMARY Chiba et al. (2006) proposed a framework of program transformation of term rewriting systems by developed templates. Contrast to the previous framework of program transformation by templates based on lambda calculus, this framework provides a method to verify the correctness of transformation automatically. Tupling (Bird, 1980) is a well-known technique to eliminate redundant recursive calls for improving efficiency of programs. In Chiba et al.'s framework, however, one can not use tuple symbols to construct developed templates. Thus their framework is not capable of tupling transformations. In this paper, we propose a more flexible notion of templates so that a wider variety of transformations, including tupling transformations, can be handled.

key words: *program transformation, tupling, term rewriting*

1. Introduction

Several techniques for optimizing functional programs by transformation have been developed [4], [9], [14], [16], [23]. One of the general such frameworks is a *program transformation by templates* proposed by Huet and Lang [16]. In this framework, a program is transformed according to a given *program transformation template*—a template consists of program schemas for input and output programs and a set of equations that expresses the precondition of operators contained in the programs. In contrast to specific program transformations such as fusion [14], [23] or tupling [3], [9], [15], the program transformation by template provides a general framework for treating various types of program transformations by preparing suitable templates in a uniform way.

In Huet and Lang's framework, the programs and program schemas are given by second-order simply-typed lambda terms. A program transformation is carried out (*automatically*) by the second-order matching and the correctness is established (*manually*) based on the denotational semantics. For the former part, matching algorithms for the program transformation by template have been improved to achieve more flexible and efficient transformations [11], [13], [20], [24]. In contrast, the latter part, namely the verification of the correctness of program transformation, has been paid less attention. But in general, the second-order matching often provides many matching solutions and not all of them are appropriate for program transformations. Therefore, the verification of the correctness is expected to be important for this framework.

Chiba et al. proposed a framework of *program transformation by templates based on term rewriting* [5]–[8]. In this framework, programs and program schemas are given by term rewriting systems (TRSs for short) and TRS patterns—a TRS in which pattern variables (which will be instantiated for program transformations) are used in the place of function symbols. They gave a criterion for the correctness of transformations by *developed templates*. The criterion consists of some properties of input and output TRSs such as confluence, sufficient completeness and of inductive validity of the instantiated precondition. The former two properties can be checked automatically in some classes and automated inductive theorem proving methods can be applied for the last property. Thus, once a suitable developed template is constructed, this result provides a method to verify the correctness of transformations automatically.

Tupling [3], [9], [15], which improves efficiency of programs by eliminating redundant recursive calls, is one of the well-known program transformation techniques in functional programming. Chiba et al.'s framework, however, is not capable of tupling, since the notion of developed templates is not expressive enough to handle tupling transformations. More precisely, one can not deal with tuple symbols, which play an essential role in tupling transformations, in the construction of developed templates. In this paper, we give a more flexible notion of templates called *correct templates* so that a wider variety of transformations, including tupling transformations, can be handled. We introduce a notion of *carrying of signatures* for term homomorphisms to give a criterion to guarantee the correctness of program transformation by those templates.

The rest of the paper is organized as follows. In Sect. 2, we first explain the framework of transformation by templates based on term rewriting and then give some basic notions and notations used in this paper. In Sect. 3, we explain tupling transformation in term rewriting. In Sect. 4, we explain a method to show the correctness of program transformation based on equivalent transformations. In Sect. 5, we introduce notions of correct templates and carrying of signatures. We then prove a new criterion to guarantee correctness of transformations. Several templates for tupling transformations are also presented. In Sect. 6, we compare our new framework with the old one. We conclude in Sect. 7.

Manuscript received July 22, 2009.
Manuscript revised November 10, 2009.

[†]The author is with Japan Advanced Institute of Science and Technology, Nomi-shi, 923–1292 Japan.
[††]The authors are with RIEC, Tohoku University, Sendai-shi, 980–8577 Japan.
a) E-mail: chiba@jaist.ac.jp
b) E-mail: aoto@nue.riec.tohoku.ac.jp
c) E-mail: toyama@nue.riec.tohoku.ac.jp

DOI: 10.1587/transinf.E93.D.963

2. Transformation by Templates

In this section, we first explain the framework of program transformation by templates based on term rewriting [5]–[7].

Let us describe a program transformation by the following template $\langle \mathcal{P}, \mathcal{P}', \mathcal{H} \rangle$.

$$\mathcal{P} \left\{ \begin{array}{lcl} \mathsf{f}(\mathsf{a}) & \to & \mathsf{b} \\ \mathsf{f}(\mathsf{c}(u_1, v_1)) & \to & \mathsf{g}(u_1, \mathsf{f}(v_1)) \\ \mathsf{g}(\mathsf{b}, u_2) & \to & u_2 \\ \mathsf{g}(\mathsf{d}(u_3, v_3), w_3) & \to & \mathsf{d}(u_3, \mathsf{g}(v_3, w_3)) \end{array} \right.$$

$$\mathcal{P}' \left\{ \begin{array}{lcl} \mathsf{f}(u_4) & \to & \mathsf{f}_1(u_4, \mathsf{b}) \\ \mathsf{f}_1(\mathsf{a}, u_5) & \to & u_5 \\ \mathsf{f}_1(\mathsf{c}(u_6, v_6), w_6) & \to & \mathsf{f}_1(v_6, \mathsf{g}(w_6, u_6)) \\ \mathsf{g}(\mathsf{b}, u_7) & \to & u_7 \\ \mathsf{g}(\mathsf{d}(u_8, v_8), w_8) & \to & \mathsf{d}(u_8, \mathsf{g}(v_8, w_8)) \end{array} \right.$$

$$\mathcal{H} \left\{ \begin{array}{lcl} \mathsf{g}(\mathsf{b}, u_1) & \approx & \mathsf{g}(u_1, \mathsf{b}) \\ \mathsf{g}(\mathsf{g}(u_2, v_2), w_2) & \approx & \mathsf{g}(u_2, \mathsf{g}(v_2, w_2)) \end{array} \right.$$

The TRS pattern $\mathcal{P}$ is the source and $\mathcal{P}'$ is the target, i.e. $\mathcal{P}$ is a schema of input programs to be transformed and $\mathcal{P}'$ is a schema whose instantiations become the output programs. The set $\mathcal{H}$ of equations is a schema of hypothesis whose instantiations are used to verify the correctness of transformations. This template is for a program transformation from recursive programs to iterative programs.

Consider, for example, the following TRS $\mathcal{R}_{\mathsf{sum}}$ which specifies a program that computes the summation of a list of natural numbers. Here the natural numbers $0, 1, 2, \ldots$ are expressed as $\mathsf{0}, \mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{s}(\mathsf{0})), \ldots$.

$$\mathcal{R}_{\mathsf{sum}} \left\{ \begin{array}{lcl} \mathsf{sum}([\,]) & \to & \mathsf{0} \\ \mathsf{sum}(x_1{:}y_1) & \to & +(x_1, \mathsf{sum}(y_1)) \\ +(\mathsf{0}, x_2) & \to & x_2 \\ +(\mathsf{s}(x_3), y_3) & \to & \mathsf{s}(+(x_3, y_3)) \end{array} \right.$$

This $\mathcal{R}_{\mathsf{sum}}$ computes the summation of a list using a recursive call. For instance, $\mathsf{sum}(1{:}(2{:}(3{:}(4{:}(5{:}[\,])))))) \xrightarrow{*}_{\mathcal{R}_{\mathsf{sum}}} +(1, +(2, +(3, +(4, +(5, \mathsf{sum}([\,])))))) \xrightarrow{*}_{\mathcal{R}_{\mathsf{sum}}} 15$.

To transform this program to the iterative form, we perform a pattern matching with the source $\mathcal{P}$ against the input TRS $\mathcal{R}_{\mathsf{sum}}$. Using the matching algorithm presented in [6], [7], one finds the following term homomorphism φ satisfying $\mathcal{R}_{\mathsf{sum}} = \varphi(\mathcal{P})$.

$$\varphi = \left\{ \begin{array}{llll} \mathsf{f} \mapsto \mathsf{sum}(\Box_1) & \mathsf{a} \mapsto [\,] & \mathsf{b} \mapsto \mathsf{0} & \\ \mathsf{g} \mapsto +(\Box_1, \Box_2) & & \mathsf{c} \mapsto \Box_1{:}\Box_2 & \\ \mathsf{f}_1 \mapsto \mathsf{sum1}(\Box_1, \Box_2) & & \mathsf{d} \mapsto \mathsf{s}(\Box_2) & \\ u_1 \mapsto x_1 & u_2 \mapsto x_2 & u_4 \mapsto x_4 & \\ u_5 \mapsto x_5 & u_6 \mapsto x_6 & u_7 \mapsto x_7 & \\ v_1 \mapsto y_1 & v_3 \mapsto x_3 & v_6 \mapsto y_6 & \\ v_8 \mapsto y_8 & w_3 \mapsto y_3 & w_6 \mapsto z_6 & \\ w_8 \mapsto z_8 & & & \end{array} \right\}$$

The output TRS $\mathcal{R}'_{\mathsf{sum}}$, the iterative form [†] of $\mathcal{R}_{\mathsf{sum}}$, is obtained by applying φ to the target $\mathcal{P}'$ i.e. $\mathcal{R}'_{\mathsf{sum}} = \varphi(\mathcal{P}')$.

$$\mathcal{R}'_{\mathsf{sum}} \left\{ \begin{array}{lcl} \mathsf{sum}(x_4) & \to & \mathsf{sum1}(x_4, \mathsf{0}) \\ \mathsf{sum1}([\,], x_5) & \to & x_5 \\ \mathsf{sum1}(x_6{:}y_6, z_6) & \to & \mathsf{sum1}(y_6, +(z_6, x_6)) \\ +(\mathsf{0}, x_7) & \to & x_7 \\ +(\mathsf{s}(y_8), z_8) & \to & \mathsf{s}(+(y_8, z_8)) \end{array} \right.$$

The term homomorphism φ is also used to generate the following set $\mathcal{E}_{\mathsf{sum}} = \varphi(\mathcal{H})$ of equations.

$$\mathcal{E}_{\mathsf{sum}} \left\{ \begin{array}{lcl} +(\mathsf{0}, x_1) & \approx & +(x_1, \mathsf{0}) \\ +(+(x_2, v_2), w_2) & \approx & +(x_2, +(v_2, w_2)) \end{array} \right.$$

It is required to ensure the correctness of the transformation that these equations are inductive consequences of $\mathcal{R}_{\mathsf{sum}}$. In the course of the transformation the inductive validity of these equations is verified, for example, by automated inductive theorem proving methods. In this case, using the rewriting induction [19], it is successfully proved automatically that the equations in $\mathcal{E}_{\mathsf{sum}}$ are inductive consequences of $\mathcal{R}_{\mathsf{sum}}$.

The correctness of the transformation, i.e. the equivalence of the input TRS $\mathcal{R}_{\mathsf{sum}}$ and the output $\mathcal{R}'_{\mathsf{sum}}$, is guaranteed if $\mathcal{R}_{\mathsf{sum}}$ is confluent and $\mathcal{R}_{\mathsf{sum}}$ and $\mathcal{R}'_{\mathsf{sum}}$ are sufficiently complete [7]—these conditions can be checked, for example, by checking the joinability of the critical pairs [2] and the complement substitution algorithm [18], [21], respectively, provided that $\mathcal{R}_{\mathsf{sum}}$ and $\mathcal{R}'_{\mathsf{sum}}$ are terminating. Termination of a TRS is an undecidable property but various termination proving methods are known and are still actively investigated. In this case, the termination of $\mathcal{R}_{\mathsf{sum}}$ and $\mathcal{R}'_{\mathsf{sum}}$ is successfully proved automatically using the lexicographic path order [2].

We now give some definitions to formalize program transformation by template based on term rewriting that will be used in this paper. Familiarity with term rewriting will be helpful in what follows (see e.g. [2]).

Let $\mathscr{F}$, $\mathscr{X}$ and $\mathscr{V}$ be the sets of *function symbols*, *pattern variables* and *local variables*, respectively. Any $p \in \mathscr{F} \cup \mathscr{X}$ has its *arity*, denoted by $\mathrm{arity}(p)$. The set $\mathrm{T}(\mathscr{F} \cup \mathscr{X}, \mathscr{V})$ of *term patterns* is defined by (1) $\mathscr{V} \subseteq \mathrm{T}(\mathscr{F} \cup \mathscr{X}, \mathscr{V})$; and (2) $p(t_1, \ldots, t_n) \subseteq \mathrm{T}(\mathscr{F} \cup \mathscr{X}, \mathscr{V})$ for any $p \in \mathscr{F} \cup \mathscr{X}$ such that $\mathrm{arity}(p) = n$ and $t_1, \ldots, t_n \in \mathrm{T}(\mathscr{F} \cup \mathscr{X}, \mathscr{V})$. A term pattern without pattern variables is called a *term*. The set of terms is denoted by $\mathrm{T}(\mathscr{F}, \mathscr{V})$. The sets of function symbols, pattern variables and local variables in a term pattern s are denoted by $\mathscr{F}(s)$, $\mathscr{X}(s)$ and $\mathscr{V}(s)$, respectively. A *ground* term (pattern) is the one without local variables. We abbreviate $\mathrm{T}(\mathscr{F} \cup \mathscr{X}, \emptyset)$, $\mathrm{T}(\mathscr{F}, \emptyset)$, etc. as $\mathrm{T}(\mathscr{F} \cup \mathscr{X})$, $\mathrm{T}(\mathscr{F})$, etc.

A *substitution* θ is a mapping from $\mathscr{V}$ to $\mathrm{T}(\mathscr{F} \cup \mathscr{X}, \mathscr{V})$. A substitution θ is extended to a mapping $\hat{\theta}$ over term pattern $\mathrm{T}(\mathscr{F} \cup \mathscr{X}, \mathscr{V})$ like this: (1) $\hat{\theta}(x) = \theta(x)$ if $x \in \mathscr{V}$, (2) $\hat{\theta}(p(s_1, \ldots, s_n)) = p(\hat{\theta}(s_1), \ldots, \hat{\theta}(s_n))$. We usually identify $\hat{\theta}$ and θ. We denote $s\theta$ instead of $\theta(s)$. The *domain* of a substitution θ (denoted by $\mathrm{dom}(\theta)$) is defined by $\mathrm{dom}(\theta) = \{x \in \mathscr{V} \mid x \neq \theta(x)\}$.

[†]Precisely speaking, only the rules for sum are changed to iterative and those for + remain recursive.

Consider special (indexed) constants $\Box_i$ ($i \geq 1$) called holes such that $\Box_i \notin \mathscr{F}$. An *(indexed) context* C is an element of $\mathrm{T}(\mathscr{F} \cup \mathscr{X} \cup \{\Box_i \mid i \geq 1\}, \mathscr{V})$. $C[s_1, \ldots, s_n]$ is the result of C replacing $\Box_i$ by $s_1, \ldots, s_n$ from left to right. $C\langle s_1, \ldots, s_n\rangle$ is the result of C replacing $\Box_i$ by s_i for $i = 1, \ldots, n$ (*indexed replacement*). (For example, $\mathsf{f}(\Box_1, \mathsf{g}(\Box_1), \Box_2)\langle x, y\rangle = \mathsf{f}(x, \mathsf{g}(x), y)$.) A context C with precisely one hole is denoted by $C[\,]$. The set of contexts is denoted by $\mathrm{T}^{\Box}(\mathscr{F} \cup \mathscr{X}, \mathscr{V})$; its subset $\mathrm{T}(\mathscr{F} \cup \mathscr{X} \cup \{\Box_i \mid 1 \leq i \leq n\}, \mathscr{V})$ is denoted by $\mathrm{T}^{\Box}_n(\mathscr{F} \cup \mathscr{X}, \mathscr{V})$. $\mathrm{T}^{\Box}(\mathscr{F})$ and $\mathrm{T}^{\Box}_n(\mathscr{F})$ are defined in the same way as $\mathrm{T}(\mathscr{F})$.

A pair $\langle l, r\rangle$ of term patterns is a *rewrite rule* if $l \notin \mathscr{V}$ and $\mathscr{V}(l) \supseteq \mathscr{V}(r)$. We usually write the rewrite rule $\langle l, r\rangle$ as $l \to r$; l is the left-hand side (lhs for short) of the rule and r is the right-hand side (rhs for short). A *term rewriting system pattern* (*TRS pattern* for short) is a set of rewrite rules. A term s *reduces* to a term t by $\mathcal{R}$ (denoted by $s \to_{\mathcal{R}} t$) if there exist a context $C[\,]$, a substitution θ and a rewrite rule $l \to r \in \mathcal{R}$ such that $s = C[l\theta]$ and $t = C[r\theta]$. The reflexive transitive closure of $\to_{\mathcal{R}}$ is denoted by $\overset{*}{\to}_{\mathcal{R}}$, the transitive closure by $\overset{+}{\to}_{\mathcal{R}}$, and the equivalence closure by $\overset{*}{\leftrightarrow}_{\mathcal{R}}$. An *equation* is a pair of term patterns; we usually write an equation $l \approx r$. Term patterns s and t are *equivalent* by a set $\mathcal{E}$ of equations (denoted by $s \leftrightarrow_{\mathcal{E}} t$) if there exist a context $C[\,]$, a substitution θ and an equation $l \approx r \in \mathcal{E}$ such that $s = C[l\theta]$ and $t = C[r\theta]$, or, $s = C[r\theta]$ and $t = C[l\theta]$. The equivalence closure of $\leftrightarrow_{\mathcal{E}}$ is denoted by $\overset{*}{\leftrightarrow}_{\mathcal{E}}$.

For a set $\Sigma \subseteq \mathscr{F} \cup \mathscr{X}$, we say a TRS pattern (a set of equations) is *over* Σ if all rewrite rules (resp. equations) consist of patterns in $\mathrm{T}(\Sigma, \mathscr{V})$. A *term rewriting system* (*TRS* for short) is a TRS pattern over $\mathscr{F}$. A rewrite rule $l \to r$ is *left-linear* if no local variable appears more than once in l. A TRS is left-linear if all rewrite rules are left-linear. A TRS $\mathcal{R}$ is *confluent*, or has the *Church-Rosser property*, ($\mathrm{CR}(\mathcal{R})$) if, for any term s, s_1, s_2, $s \overset{*}{\to}_{\mathcal{R}} s_1$ and $s \overset{*}{\to}_{\mathcal{R}} s_2$ imply that there exists a term t such that $s_1 \overset{*}{\to}_{\mathcal{R}} t$ and $s_2 \overset{*}{\to}_{\mathcal{R}} t$. A TRS $\mathcal{R}$ is *strongly normalizing* or *terminating* ($\mathrm{SN}(\mathcal{R})$) if there exists no infinite reduction $s_1 \to_{\mathcal{R}} s_2 \to_{\mathcal{R}} s_3 \to_{\mathcal{R}} \cdots$. We note that confluence and termination of TRSs are undecidable problems in general. However, confluence of terminating TRSs can be decided by checking the joinability of critical pairs of $\mathcal{R}$ [2].

We assume that the set $\mathscr{F}$ of function symbols is divided into two disjoint sets—the set $\mathscr{F}_{\mathrm{d}}$ of *defined function symbols* and the set $\mathscr{F}_{\mathrm{c}}$ of *constructor symbols*. A rewrite rule $l \to r$ is a *constructor rewrite rule* if $l = f(l_1, \ldots, l_n)$ for some $f \in \mathscr{F}_{\mathrm{d}}$ and $l_1, \ldots, l_n \in \mathrm{T}(\mathscr{F}_{\mathrm{c}}, \mathscr{V})$. A TRS $\mathcal{R}$ is a *constructor system* (*CS* for short) if all rewrite rules are constructor rewrite rules.

Suppose $\mathscr{F}_{\mathrm{c}} \subseteq \mathscr{G} \subseteq \mathscr{F}$. A substitution θ is *ground on* $\mathscr{G}$ if $\theta(x) \in \mathrm{T}(\mathscr{G})$ for any $x \in \mathrm{dom}(\theta)$. An equation $s \approx t$ is an *inductive consequence of* $\mathcal{R}$ *for* $\mathscr{G}$, or *inductively valid in* $\mathcal{R}$ *for* $\mathscr{G}$, ($\mathcal{R}, \mathscr{G} \vdash_{ind} s \approx t$) if for any ground substitution θ_g on $\mathscr{G}$ such that $\mathscr{V}(s) \cup \mathscr{V}(t) \subseteq \mathrm{dom}(\theta_g)$, $s\theta_g \overset{*}{\leftrightarrow}_{\mathcal{R}} t\theta_g$ holds. For a set $\mathcal{E}$ of equations, we write $\mathcal{R}, \mathscr{G} \vdash_{ind} \mathcal{E}$ when $\mathcal{R}, \mathscr{G} \vdash_{ind} s \approx t$ for any $s \approx t \in \mathcal{E}$. It is undecidable in general that whether an equation is an inductive consequence of a TRS. There are, however, several automated methods, such as the rewriting induction [19], for proving/disproving inductive validity are known.

A TRS $\mathcal{R}$ is *sufficiently complete for* $\mathscr{G}$ ($\mathrm{SC}(\mathcal{R}, \mathscr{G})$) if for any ground term $s \in \mathrm{T}(\mathscr{G})$ there exists $t \in \mathrm{T}(\mathscr{F}_{\mathrm{c}})$ such that $s \overset{*}{\to}_{\mathcal{R}} t$. Sufficient completeness of TRSs is an undecidable problem. However, sufficient completeness of terminating TRSs can be decided by the complement substitution algorithm [18], [21]. Furthermore, for many TRSs suitable for programs, sufficient completeness of TRSs does not hold if one does not consider many-sorted signature. In many-sorted signature, function symbols are equipped with not only the arities but with sorted specification $\alpha_1 \times \cdots \times \alpha_n \to \alpha_0$ for arguments and the return value where $\alpha_0, \alpha_1, \ldots, \alpha_n$ are sorts. For example, considering the sort Nat for the set of natural numbers and the sort List for the set of lists of natural numbers, the function symbol + is assigned by $\mathsf{Nat} \times \mathsf{Nat} \to \mathsf{Nat}$ ($+ : \mathsf{Nat} \times \mathsf{Nat} \to \mathsf{Nat}$) and the function symbol sum is assigned by $\mathsf{List} \to \mathsf{Nat}$ ($\mathsf{sum} : \mathsf{List} \to \mathsf{Nat}$). However, we proceed [†] our discussion in the mono-sorted framework; for, the discussion of this paper can be extended to the many-sorted framework in the straightforward way.

A *transformation template* (or just *template*) is a triple $\langle \mathcal{P}, \mathcal{P}', \mathcal{H}\rangle$ of TRS patterns $\mathcal{P}, \mathcal{P}'$ and a set $\mathcal{H}$ of equations. The TRS patterns $\mathcal{P}$ and $\mathcal{P}'$ are the *source* and the *target* of the template, respectively, and the set $\mathcal{H}$ of equations is the *hypothesis* of the template. A mapping φ from $\mathscr{X} \cup \mathscr{V}$ to $\mathrm{T}^{\Box}(\mathscr{F} \cup \mathscr{X}, \mathscr{V})$ is said to be a *term homomorphism* if (1) $\varphi(p) \in \mathrm{T}^{\Box}_{\mathrm{arity}(p)}(\mathscr{F} \cup \mathscr{X})$ for any $p \in \mathrm{dom}_{\mathscr{X}}(\varphi)$, (2) $\varphi(x) \in \mathscr{V}$ for any $x \in \mathrm{dom}_{\mathscr{V}}(\varphi)$, and (3) φ is injective on $\mathrm{dom}_{\mathscr{V}}(\varphi)$, i.e., for any $x, y \in \mathrm{dom}_{\mathscr{V}}(\varphi)$, if $x \neq y$ then $\varphi(x) \neq \varphi(y)$. Here, the *domains of* φ *over pattern variables and local variables* are defined by $\mathrm{dom}_{\mathscr{X}}(\varphi) = \{p \in \mathscr{X} \mid \varphi(p) \neq p(\Box_1, \ldots, \Box_{\mathrm{arity}(p)})\}$ and $\mathrm{dom}_{\mathscr{V}}(\varphi) = \{x \in \mathscr{V} \mid \varphi(x) \neq x\}$, respectively. A term homomorphism φ is extended to a mapping φ over $\mathrm{T}(\mathscr{F} \cup \mathscr{X}, \mathscr{V})$ as follows:

$$\varphi(s) = \begin{cases} \varphi(x) & \text{if } s = x \in \mathscr{V} \\ f(\varphi(s_1), \ldots, \varphi(s_n)) & \\ \qquad \text{if } s = f(s_1, \ldots, s_n),\ f \in \mathscr{F} & \\ \varphi(p)\langle\varphi(s_1), \ldots, \varphi(s_n)\rangle & \\ \qquad \text{if } s = p(s_1, \ldots, s_n),\ p \in \mathscr{X}. & \end{cases}$$

A term homomorphism is extended to a mapping on rewrite rules and equations in the obvious way. For term homomorphisms φ and φ' such that $\mathrm{dom}(\varphi) \cap \mathrm{dom}(\varphi') = \emptyset$, $\varphi \cup \varphi'$ is a term homomorphism defined by $(\varphi \cup \varphi')(p)$ equals $\varphi'(p)$ if $p \in \mathrm{dom}(\varphi')$ and $\varphi(p)$ otherwise. The following basic property of the term homomorphisms will be used later.

Proposition 1 (Proposition 1 of [6]): Let φ be a term homomorphism, $\mathcal{P}$ a TRS pattern and $\mathcal{H}$ a set of equations. If $s \to_{\mathcal{P}} t$ ($s \leftrightarrow_{\mathcal{H}} t$) then we have $\varphi(s) \to_{\varphi(\mathcal{P})} \varphi(t)$ (resp. $\varphi(s) \leftrightarrow_{\varphi(\mathcal{H})} \varphi(t)$).

[†]This is very usual (see e.g. Sect. 3 of [2]).

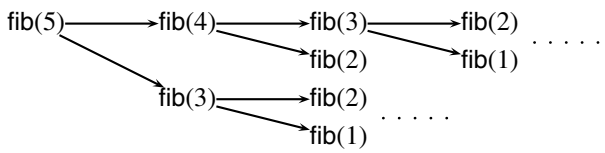
3. Tupling Transformations for Term Rewriting Systems

Tupling is one of the well-known program transformation methods in the field of functional programming [3], [9], [10], [15]. In this section, we give some examples of tupling transformations in term rewriting.

Let us consider one source of inefficiency in programs—namely, inefficient programs may perform the same computations while efficient programs share this same computations and use the result many times. Such a typical program arises from a usual definition of Fibonacci number.

$$\mathcal{R}_{\mathsf{fib}}\left\{\begin{array}{lcl}\mathsf{fib}(\mathsf{0}) & \to & \mathsf{s}(\mathsf{0})\\ \mathsf{fib}(\mathsf{s}(\mathsf{0})) & \to & \mathsf{s}(\mathsf{0})\\ \mathsf{fib}(\mathsf{s}(\mathsf{s}(x))) & \to & +(\mathsf{fib}(\mathsf{s}(x)),\mathsf{fib}(x))\\ +(\mathsf{s}(x),y) & \to & \mathsf{s}(+(x,y))\\ +(\mathsf{0},x) & \to & x\end{array}\right.$$

In this program, the function calls from fib(5) are described in a directed graph as follows.



We see that fib(3) is computed twice and fib(2) is computed three times and these computations are performed independently. Such multiple computations of the same expression are the reason of exponential time computation of fib(n).

Tupling eliminates such multiple computations of the same expression called in different contexts. One of the basic methods of tupling in functional programs is operated as follows [9]. First one needs to find some suitable expressions that should be computed in simultaneously. Then one prepares a new function definition which computes these expressions together. Finally, one makes original functions derived from this new function. The name of "tupling" comes from the tuple symbols "$\langle\cdot\rangle$" used for coupling the expressions that are simultaneously computed.

In our example, by tupling, one gets the following $\mathcal{R}'_{\mathsf{fib}}$ from $\mathcal{R}_{\mathsf{fib}}$.

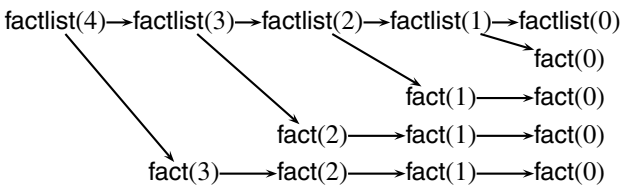
$$\mathcal{R}'_{\mathsf{fib}}\left\{\begin{array}{lcl}\mathsf{fib}(\mathsf{0}) & \to & \mathsf{s}(\mathsf{0})\\ \mathsf{fib}(\mathsf{s}(\mathsf{0})) & \to & \mathsf{s}(\mathsf{0})\\ \mathsf{fib}(\mathsf{s}(\mathsf{s}(x))) & \to & \pi_1(\mathsf{step}(\mathsf{fibpair}(x)))\\ \mathsf{fibpair}(\mathsf{0}) & \to & \langle\mathsf{s}(\mathsf{0}),\mathsf{s}(\mathsf{0})\rangle\\ \mathsf{fibpair}(\mathsf{s}(x)) & \to & \mathsf{step}(\mathsf{fibpair}(x))\\ \mathsf{step}(x) & \to & \langle+(\pi_1(x),\pi_2(x)),\pi_1(x)\rangle\\ +(\mathsf{0},x) & \to & x\\ +(\mathsf{s}(x),y) & \to & \mathsf{s}(+(x,y))\\ \pi_1(\langle x,y\rangle) & \to & x\\ \pi_2(\langle x,y\rangle) & \to & y\end{array}\right.$$

In $\mathcal{R}'_{\mathsf{fib}}$, fibpair($n$) computes fib($n$) and fib($n-1$) simultaneously. step($\langle x,y\rangle$) computes $\langle x+y,x\rangle$. If x,y are values then this can be computed efficiently. In term rewriting, this is done by sharing evaluation of x in rhs of the rule $\mathsf{step}(x)\to\langle+(\pi_1(x),\pi_2(x)),\pi_1(x)\rangle$ (based on e.g. term graph rewriting framework). Thus, fib(n) can be computed much efficiently using $\mathcal{R}'_{\mathsf{fib}}$ instead of $\mathcal{R}_{\mathsf{fib}}$.

Let us see another example of tupling transformation. In the following TRS $\mathcal{R}_{\mathsf{factlist}}$, factlist($n$) computes a list $[(n-1)!,\ldots,0!]$.

$$\mathcal{R}_{\mathsf{factlist}}\left\{\begin{array}{lcl}\mathsf{factlist}(\mathsf{0}) & \to & [\,]\\ \mathsf{factlist}(\mathsf{s}(x)) & \to & \mathsf{fact}(x){:}\mathsf{factlist}(x)\\ \mathsf{fact}(\mathsf{0}) & \to & \mathsf{s}(\mathsf{0})\\ \mathsf{fact}(\mathsf{s}(x)) & \to & \times(\mathsf{s}(x),\mathsf{fact}(x))\\ \times(\mathsf{0},y) & \to & \mathsf{0}\\ \times(\mathsf{s}(x),y) & \to & +(y,\times(x,y))\\ +(\mathsf{0},x) & \to & x\\ +(\mathsf{s}(x),y) & \to & \mathsf{s}(+(x,y))\end{array}\right.$$

The dependency of function calls from factlist(4) can be described by the following graph:



We see that fact($n-1$) called from fact(n) and factlist(n) can be shared. By a tupling transformation, we get the following TRS $\mathcal{R}'_{\mathsf{factlist}}$ from $\mathcal{R}_{\mathsf{factlist}}$.

$$\mathcal{R}'_{\mathsf{factlist}}\left\{\begin{array}{lcl}\mathsf{factlist}(x) & \to & \pi_1(\mathsf{factpair}(x))\\ \mathsf{fact}(x) & \to & \pi_2(\mathsf{factpair}(x))\\ \mathsf{factpair}(\mathsf{0}) & \to & \langle[\,],\mathsf{s}(\mathsf{0})\rangle\\ \mathsf{factpair}(\mathsf{s}(x)) & \to & \mathsf{step}(\mathsf{s}(x),\mathsf{factpair}(x))\\ \mathsf{step}(x,y) & \to & \langle\pi_2(y){:}\pi_1(y),\times(x,\pi_2(y))\rangle\\ \times(\mathsf{0},y) & \to & \mathsf{0}\\ \times(\mathsf{s}(x),y) & \to & +(y,\times(x,y))\\ +(\mathsf{0},x) & \to & x\\ +(\mathsf{s}(x),y) & \to & \mathsf{s}(+(x,y))\\ \pi_1(\langle x,y\rangle) & \to & x\\ \pi_2(\langle x,y\rangle) & \to & y\end{array}\right.$$

In $\mathcal{R}'_{\mathsf{factlist}}$, factpair($n$) computes factlist($n$) and fact($n$) simultaneously. step($x,\langle ys,y\rangle$) computes $\langle y{:}ys,x\times y\rangle$. If x,y,ys are values then this can be computed efficiently.

4. Correctness of Program Transformation

A first question that arises is how correctness of such program transformation can be verified. In this section, we explain a method based on the equivalent transformation [6], [7], [22] on which our framework is based.

To discuss correctness of program transformations, we need to argue about the notion of equivalence of TRSs suitable for program transformations. Let $\mathscr{G}$ be a set of function symbols such that $\mathscr{F}_{\mathrm{c}}\subseteq\mathscr{G}\subseteq\mathscr{F}$. Two TRSs $\mathcal{R}$ and $\mathcal{R}'$ are said to be *equivalent for* $\mathscr{G}$ ($\mathcal{R}\simeq_{\mathscr{G}}\mathcal{R}'$), if for any

$s \in \mathrm{T}(\mathscr{G})$ and $t \in \mathrm{T}(\mathscr{F}_c)$, $s \overset{*}{\to}_{\mathcal{R}} t$ iff $s \overset{*}{\to}_{\mathcal{R}'} t$ holds [6], [7]. We note that program transformation may introduce auxiliary function symbols such as sum1, factpair, etc. This is why one has to restrict his/her attention to some set $\mathscr{G}$ of function symbols. *Equivalent transformation* [6], [7], [22] can be used to establish the equivalence of two TRSs.

Definition 1: Let $\mathcal{R}_0$ be a left-linear CS over $\mathscr{F}_0$ and $\mathcal{E}$ a set of equations over $\mathscr{F}_0$. An *equivalent transformation sequence under* $\mathcal{E}$ is a sequence $\mathcal{R}_0, \ldots, \mathcal{R}_n$ of TRSs (over $\mathscr{F}_0, \ldots, \mathscr{F}_n$, respectively) such that $\mathcal{R}_{k+1}$ is obtained from $\mathcal{R}_k$ by applying one of the following inference rules:

(I) *Introduction*

$$\mathcal{R}_{k+1} = \mathcal{R}_k \cup \{f(x_1, \ldots, x_n) \to r\}$$

provided that $f \notin \mathscr{F}_k$, and $r \in \mathrm{T}(\mathscr{F}_k, \{x_1, \ldots, x_n\})$, where $x_1, \ldots, x_n$ are mutually distinct variables. We put $\mathscr{F}_{k+1} = \mathscr{F}_k \cup \{f\}$.

(A) *Addition*

$$\mathcal{R}_{k+1} = \mathcal{R}_k \cup \{l \to r\}$$

provided $l \overset{*}{\leftrightarrow}_{\mathcal{R}_k \cup \mathcal{E}} r$ holds.

(E) *Elimination*

$$\mathcal{R}_{k+1} = \mathcal{R}_k \setminus \{l \to r\}$$

If this is the case, we write $\mathcal{R}_k \Rightarrow \mathcal{R}_{k+1}$. (In the Addition and Elimination rules, $\mathscr{F}_{k+1}$ can be any set of function symbols such that $\mathscr{F}_{k+1} \subseteq \mathscr{F}_k$ provided that $\mathcal{R}_{k+1}$ is a TRS over $\mathscr{F}_{k+1}$.) The reflexive transitive closure of $\Rightarrow$ is denoted by $\overset{*}{\Rightarrow}$. We indicate the rule of $\Rightarrow$ by $\underset{I}{\Rightarrow}$, $\underset{A}{\Rightarrow}$, or $\underset{E}{\Rightarrow}$. Finally, we say there exists an *equivalent transformation from* $\mathcal{R}$ *to* $\mathcal{R}'$ *under* $\mathcal{E}$ if there exists an equivalent transformation sequence $\mathcal{R} \overset{*}{\underset{I}{\Rightarrow}} \cdot \overset{*}{\underset{A}{\Rightarrow}} \cdot \overset{*}{\underset{E}{\Rightarrow}} \mathcal{R}'$ under $\mathcal{E}$.

The next proposition is the basis of correctness of program transformation by template based on term rewriting.

Proposition 2 (Theorem 4.4 of [7]): Let $\mathscr{G}$ and $\mathscr{G}'$ be sets of function symbols such that $\mathscr{F}_c \subseteq \mathscr{G}, \mathscr{G}' \subseteq \mathscr{F}$. Let $\mathcal{R}$ be a left-linear CS over $\mathscr{G}$, $\mathcal{E}$ a set of equations over $\mathscr{G}$, and $\mathcal{R}'$ a TRS over $\mathscr{G}'$. Suppose that $\mathcal{R}, \mathscr{G} \vdash_{ind} \mathcal{E}$ and there exists an equivalent transformation from $\mathcal{R}$ to $\mathcal{R}'$ under $\mathcal{E}$. Then, $\mathrm{CR}(\mathcal{R}) \wedge \mathrm{SC}(\mathcal{R}, \mathscr{G}) \wedge \mathrm{SC}(\mathcal{R}', \mathscr{G}')$ imply $\mathcal{R} \simeq_{\mathscr{G} \cap \mathscr{G}'} \mathcal{R}'$.

The transformation from $\mathcal{R}_{\mathsf{factlist}}$ to $\mathcal{R}'_{\mathsf{factlist}}$ in the previous section is based on the tupling transformation of [9]. We are now going to demonstrate how the same transformation is obtained by the method of equivalent transformations (Definition 1) and how the correctness of the transformation is obtained by Proposition 2.

First, we show an equivalent transformation from $\mathcal{R}_{\mathsf{factlist}} \cup \mathcal{R}_\pi$ to $\mathcal{R}'_{\mathsf{factlist}}$ under the empty set $\emptyset$, where $\mathcal{R}_\pi = \{\pi_1(\langle x, y\rangle) \to x, \pi_2(\langle x, y\rangle) \to y\}$.

1. Let $\mathcal{R}_0 = \mathcal{R}_{\mathsf{factlist}} \cup \mathcal{R}_\pi$.
2. Let $\mathcal{R}_1 = \mathcal{R}_0 \cup \{\mathsf{factpair}(x) \to \langle \mathsf{factlist}(x), \mathsf{fact}(x)\rangle\}$. Here, factpair is a fresh function symbol. Thus, $\mathcal{R}_0 \Rightarrow \mathcal{R}_1$ by the Introduction rule.
3. Let $\mathcal{R}_2 = \mathcal{R}_1 \cup \{\mathsf{step}(x, y) \to \langle \pi_1(y){:}\pi_2(y), x \times \pi_2(y)\rangle\}$. Here, step is a fresh function symbol. Thus, $\mathcal{R}_1 \Rightarrow \mathcal{R}_2$ by the Introduction rule.
4. Let $\mathcal{R}_3 = \mathcal{R}_2 \cup \{\mathsf{factpair}(0) \to \langle [\,], \mathsf{s}(0)\rangle\}$. Here, we have $\mathsf{factpair}(0) \to_{\mathcal{R}_2} \langle \mathsf{factlist}(0), \mathsf{fact}(0)\rangle \overset{*}{\to}_{\mathcal{R}_2} \langle [\,], \mathsf{s}(0)\rangle$. Thus, $\mathcal{R}_2 \Rightarrow \mathcal{R}_3$ by the Addition rule.
5. Let $\mathcal{R}_4 = \mathcal{R}_3 \cup \{\mathsf{factpair}(\mathsf{s}(x)) \to \mathsf{step}(\mathsf{s}(x), \mathsf{factpair}(x))\}$. Here, we have $\mathsf{factpair}(\mathsf{s}(x))$

$$\begin{aligned}
&\to_{\mathcal{R}_3} \langle \mathsf{factlist}(\mathsf{s}(x)), \mathsf{fact}(\mathsf{s}(x))\rangle \\
&\overset{*}{\to}_{\mathcal{R}_3} \langle \mathsf{fact}(x){:}\mathsf{factlist}(x), \mathsf{s}(x){\times}\mathsf{fact}(x)\rangle \\
&\overset{*}{\leftarrow}_{\mathcal{R}_3} \langle \pi_2(\langle \mathsf{factlist}(x), \mathsf{fact}(x)\rangle) \\
&\qquad\qquad :\pi_1(\langle \mathsf{factlist}(x), \mathsf{fact}(x)\rangle), \\
&\qquad\quad \mathsf{s}(x){\times}\pi_2(\langle \mathsf{factlist}(x), \mathsf{fact}(x)\rangle)\rangle \\
&\overset{*}{\leftarrow}_{\mathcal{R}_3} \langle \pi_2(\mathsf{factpair}(x)){:}\pi_1(\mathsf{factpair}(x)), \\
&\qquad\quad \mathsf{s}(x){\times}\pi_2(\mathsf{factpair}(x))\rangle \\
&\leftarrow_{\mathcal{R}_3} \mathsf{step}(\mathsf{s}(x), \mathsf{factpair}(x))
\end{aligned}$$

Thus, $\mathcal{R}_3 \Rightarrow \mathcal{R}_4$ by the Addition rule.
6. Let $\mathcal{R}_5 = \mathcal{R}_4 \cup \{\mathsf{factlist}(x) \to \pi_1(\mathsf{factpair}(x))\}$. Here, we have $\mathsf{factlist}(x) \leftarrow_{\mathcal{R}_4} \pi_1(\langle \mathsf{factlist}(x), \mathsf{fact}(x)\rangle) \leftarrow_{\mathcal{R}_4} \pi_1(\mathsf{factpair}(x))$. Thus, $\mathcal{R}_4 \Rightarrow \mathcal{R}_5$ by the Addition rule.
7. Let $\mathcal{R}_6 = \mathcal{R}_5 \cup \{\mathsf{fact}(x) \to \pi_2(\mathsf{factpair}(x))\}$. Here, we have $\mathsf{fact}(x) \leftarrow_{\mathcal{R}_5} \pi_2(\langle \mathsf{factlist}(x), \mathsf{fact}(x)\rangle) \leftarrow_{\mathcal{R}_5} \pi_2(\mathsf{factpair}(x))$. Thus, $\mathcal{R}_5 \Rightarrow \mathcal{R}_6$ by the Addition rule.
8. Finally, applying the Elimination rule five times to $\mathcal{R}_6$, we obtain $\mathcal{R}'_{\mathsf{factlist}}$.

Thus $\mathcal{R}_{\mathsf{factlist}} \cup \mathcal{R}_\pi \overset{*}{\underset{I}{\Rightarrow}} \cdot \overset{*}{\underset{A}{\Rightarrow}} \cdot \overset{*}{\underset{E}{\Rightarrow}} \mathcal{R}'_{\mathsf{factlist}}$ under $\emptyset$, i.e. there is an equivalent transformation from $\mathcal{R}_{\mathsf{factlist}} \cup \mathcal{R}_\pi$ to $\mathcal{R}'_{\mathsf{factlist}}$ under $\emptyset$.

Next, we show the correctness of the transformation based on Proposition 2. For this, we need to consider [†] the many-sorted signature additionally to guarantee the sufficient completeness. We consider the following natural many-sorted signature $\mathsf{factlist} : \mathsf{Nat} \to \mathsf{List}$, $\mathsf{fact} : \mathsf{Nat} \to \mathsf{Nat}$, $\times : \mathsf{Nat} \times \mathsf{Nat} \to \mathsf{Nat}$, $+ : \mathsf{Nat} \times \mathsf{Nat} \to \mathsf{Nat}$, $: : \mathsf{Nat} \times \mathsf{List} \to \mathsf{List}$, $[\,] : \mathsf{List}$, $\mathsf{s} : \mathsf{Nat} \to \mathsf{Nat}$, $0 : \mathsf{Nat}$, $\langle\cdot\rangle : \mathsf{List} \times \mathsf{Nat} \to \mathsf{Pair}$, $\pi_1 : \mathsf{Pair} \to \mathsf{List}$, $\pi_2 : \mathsf{Pair} \to \mathsf{Nat}$, $\mathsf{factpair} : \mathsf{Nat} \to \mathsf{Pair}$, and $\mathsf{step} : \mathsf{Nat} \times \mathsf{Pair} \to \mathsf{Pair}$.

We now show $\mathcal{R}_{\mathsf{factlist}} \cup \mathcal{R}_\pi \simeq_{\mathscr{G} \cap \mathscr{G}'} \mathcal{R}'_{\mathsf{factlist}}$ where $\mathscr{G} = \{\mathsf{factlist}, \mathsf{fact}, \times, +, :, [\,], \mathsf{s}, 0, \pi_1, \pi_2, \langle\cdot\rangle\}$ and $\mathscr{G}' = \mathscr{G} \cup \{\mathsf{factpair}, \mathsf{step}\}$. Since there is an equivalent transformation from $\mathcal{R}_{\mathsf{factlist}} \cup \mathcal{R}_\pi$ to $\mathcal{R}'_{\mathsf{factlist}}$ under $\emptyset$, it suffices to check the rest of the conditions of Proposition 2.

It is easy to see that $\mathcal{R}_{\mathsf{factlist}} \cup \mathcal{R}_\pi$ is a left-linear CS. Since $\mathcal{E} = \emptyset$, $\mathcal{R}, \mathscr{G} \vdash_{ind} \mathcal{E}$ holds trivially. To show $\mathrm{CR}(\mathcal{R}_{\mathsf{factlist}} \cup \mathcal{R}_\pi)$, $\mathrm{SC}(\mathcal{R}_{\mathsf{factlist}} \cup \mathcal{R}_\pi, \mathscr{G})$ and $\mathrm{SC}(\mathcal{R}'_{\mathsf{factlist}}, \mathscr{G}')$, we use the fact $\mathrm{SN}(\mathcal{R}_{\mathsf{factlist}} \cup \mathcal{R}_\pi)$ and $\mathrm{SN}(\mathcal{R}'_{\mathsf{factlist}})$—they can be shown automatically using the lexicographic path order. Then $\mathrm{CR}(\mathcal{R}_{\mathsf{factlist}} \cup \mathcal{R}_\pi)$ follows since there is no crit-

[†] Recall a remark after the definition of sufficient completeness in Sect. 2.

ical pairs in $\mathcal{R}_{\mathsf{factlist}} \cup \mathcal{R}_\pi$. Furthermore, based on the many-sorted assumption, it is easy to check $\mathrm{SC}(\mathcal{R}_{\mathsf{factlist}} \cup \mathcal{R}_\pi, \mathscr{G})$ and $\mathrm{SC}(\mathcal{R}'_{\mathsf{factlist}}, \mathscr{G}')$ using the complement substitution algorithm. This complete the proof of the equivalence $\mathcal{R}_{\mathsf{factlist}} \cup \mathcal{R}_\pi \simeq_{\mathscr{G}\cap\mathscr{G}'} \mathcal{R}'_{\mathsf{factlist}}$.

As demonstrated above, once termination of a TRS has been proved, confluence and sufficient completeness of the TRS can be decided. Although termination is an undecidable property, a number of techniques are available to show termination of TRSs. There are also several automated methods for proving/disproving whether an equation is an inductive consequence of a TRS. In contrast, as one may expect, it is difficult to automate the discovery of a suitable set $\mathcal{E}$ of equations and an equivalent transformation sequence from $\mathcal{R}$ to $\mathcal{R}'$ under $\mathcal{E}$, from given TRSs $\mathcal{R}$ and $\mathcal{R}'$ only. Thus this framework is expected to be used only when one is working with his/her hands.

5. Extended Templates and Its Correctness

Since it is difficult to find a suitable equivalent transformation sequence automatically for each program transformation, the framework in the previous section is not directly applicable to *automated* verification of the correctness of program transformations.

In the framework of program transformation by templates, this can be partially achieved by abstracting the similar equivalent transformations of TRSs into a transformation of templates. The idea is to prepare a template constructed by an abstract equivalent transformation and to lift up from an abstract equivalent transformation $\mathcal{P} \overset{*}{\Rightarrow} \mathcal{P}'$ to a concrete equivalent transformation $\varphi(\mathcal{P}) \overset{*}{\Rightarrow} \varphi(\mathcal{P}')$.

To make such a lift up possible, the notions of *developed templates* and *CS homomorphisms* have been introduced in [5]–[8]. As explained later, however, these notions turn out to be too restrictive to deal with tupling transformations, notwithstanding the successful tupling transformation based on equivalent transformation in the previous section. This motivates us to extend the notion of developed templates. We first give an equivalent transformation of templates which extends the one in [5]–[8].

Definition 2: Let $\mathcal{P}_0$ be a TRS pattern over a set $\Sigma_0 \subseteq \mathscr{F} \cup \mathscr{X}$ and $\mathcal{H}$ a set of equations over Σ_0. An *abstract equivalent transformation sequence under* $\mathcal{H}$ is a sequence $\mathcal{P}_0, \ldots, \mathcal{P}_n$ of TRS patterns (over $\Sigma_0, \ldots, \Sigma_n$, respectively) such that $\mathcal{P}_{k+1}$ is obtained from $\mathcal{P}_k$ by applying one of the following inference rules:

(I) *Introduction*

$$\mathcal{P}_{k+1} = \mathcal{P}_k \cup \{p(x_1, \ldots, x_n) \to r\}$$

provided that $p \in \mathscr{X} \setminus \Sigma_k$ and $r \in \mathrm{T}(\Sigma_k, \{x_1, \ldots, x_n\})$, where $x_1, \ldots, x_n$ are mutually distinct variables. We put $\Sigma_{k+1} = \Sigma_k \cup \{p\}$.

(A) *Addition*

$$\mathcal{P}_{k+1} = \mathcal{P}_k \cup \{l \to r\}$$

provided $l \overset{*}{\leftrightarrow}_{\mathcal{P}_k \cup \mathcal{H}} r$ holds.

(E) *Elimination*

$$\mathcal{P}_{k+1} = \mathcal{P}_k \setminus \{l \to r\}$$

If this is the case, we write $\mathcal{P}_k \Rightarrow \mathcal{P}_{k+1}$. (In the Addition and Elimination rules, Σ_{k+1} can be any set such that $\Sigma_{k+1} \subseteq \Sigma_k$ provided that $\mathcal{P}_{k+1}$ is a TRS pattern over Σ_{k+1}.) The reflexive transitive closure of $\Rightarrow$ is denoted by $\overset{*}{\Rightarrow}$. We indicate the rule of $\Rightarrow$ by $\underset{I}{\Rightarrow}$, $\underset{A}{\Rightarrow}$, or $\underset{E}{\Rightarrow}$. Finally, we say that $\langle \mathcal{P}, \mathcal{P}', \mathcal{H} \rangle$ is a *correct template* if there exists an abstract equivalent transformation sequence $\mathcal{P} \overset{*}{\underset{I}{\Rightarrow}} \cdot \overset{*}{\underset{A}{\Rightarrow}} \cdot \overset{*}{\underset{E}{\Rightarrow}} \mathcal{P}'$ under $\mathcal{H}$.

Unfortunately, the existence of abstract equivalent transformation $\mathcal{P} \overset{*}{\Rightarrow} \mathcal{P}'$ under $\mathcal{H}$ does not imply that of equivalent transformation $\varphi(\mathcal{P}) \overset{*}{\Rightarrow} \varphi(\mathcal{P}')$ under $\varphi(\mathcal{H})$ for any term homomorphism φ. More specifically, the application of Introduction rule is not preserved. For example, $\varphi(p(x_1, \ldots, x_n)) \to \varphi(r)$ may be (1) $\mathsf{f}(x_1) \to \mathsf{g}(x_1, x_2)$, (2) $\mathsf{g}(x_1, x_1) \to r$, (3) $\mathsf{f}(\mathsf{f}(x_1)) \to r$, or (4) $x_1 \to r$, all of which do not satisfy the conditions of Introduction rule for an equivalent transformation. Namely, for (1), the variable condition ($\mathcal{V}(r) \subseteq \mathcal{V}(l)$) is violated; for (2), lhs is not linear; for (3), the proper subterm of lhs is not a variable; and for (4), the root symbol of lhs is not a function symbol. Similarly, one needs to guarantee $f \notin \mathscr{F}_k$ and $\mathscr{F}(r) \subseteq \mathscr{F}_k$. Therefore, in order to guarantee the successful lift up of abstract equivalent transformation sequences, one has to impose some conditions on the term homomorphism φ to use. The following condition of term homomorphisms is helpful to specify the needed conditions.

Definition 3: Let $\Sigma \subseteq \mathscr{F} \cup \mathscr{X}$ and $\mathscr{G} \subseteq \mathscr{F}$. A term homomorphism φ *carries* Σ *to* $\mathscr{G}$ if $\Sigma \cap \mathscr{F} \subseteq \mathscr{G}$ and $\varphi(p) \in \mathrm{T}(\mathscr{G} \cup \mathscr{H}_{\mathrm{arity}(p)})$ for all $p \in \Sigma \cap \mathscr{X}$.

Note that it immediately follows from this definition that $s \in \mathrm{T}(\Sigma, \mathscr{V})$ implies $\varphi(s) \in \mathrm{T}(\mathscr{G}, \mathscr{V})$ for any term homomorphism φ that carries Σ to $\mathscr{G}$. We also note that, for term homomorphism φ with a finite domain, it is decidable whether φ carries Σ to $\mathscr{G}$.

Lemma 1: Let $\Sigma \subseteq \mathscr{F} \cup \mathscr{X}$ and $\mathcal{P}$ and $\mathcal{H}$ be a TRS pattern and a set of equations over Σ, respectively. Suppose $\Sigma' \subseteq \mathscr{F} \cup \mathscr{X}$, $\mathcal{P}'$ is a TRS pattern over Σ', and that there is an abstract equivalent transformation sequence under $\mathcal{H}$ from $\mathcal{P}$ to $\mathcal{P}'$. Let $\mathscr{G} \subseteq \mathscr{F}$ and φ a term homomorphism such that (1) φ carries Σ to $\mathscr{G}$, (2) $\varphi(\mathcal{P})$ is a left-linear CS, and (3) for each $p \in \mathscr{X} \setminus \Sigma$, $\varphi(p) = f_p(\Box_1, \ldots, \Box_{\mathrm{arity}(p)})$ with mutually distinct $f_p \in \mathscr{F}_{\mathrm{d}} \setminus \Sigma$. Then there exists an equivalent transformation sequence $\varphi(\mathcal{P}) \overset{*}{\Rightarrow} \varphi(\mathcal{P}')$ under $\varphi(\mathcal{H})$.

Proof. We lift up the abstract equivalent transformation sequence $\mathcal{P} = \mathcal{P}_0 \Rightarrow \cdots \Rightarrow \mathcal{P}_n = \mathcal{P}'$ (over $\Sigma = \Sigma_0, \ldots, \Sigma_n = \Sigma'$, respectively) to the equivalent transformation sequence $\varphi(\mathcal{P}) = \varphi(\mathcal{P}_0) \Rightarrow \cdots \Rightarrow \varphi(\mathcal{P}_n) = \varphi(\mathcal{P}')$ (over $\mathscr{G} = \mathscr{F}_0, \ldots, \mathscr{F}_n$, respectively) using the corresponding transformation rules. For the transformation $\mathcal{P}_0 \overset{*}{\underset{I}{\Rightarrow}} \mathcal{P}_k$,

we have $\Sigma_{i+1} = \Sigma_i \cup \{p_{i+1}\}$ $(0 \leq i < k)$, for some $p_{i+1} \notin \Sigma_i$. For each i, we put $\mathscr{F}_{i+1} = \mathscr{F}_i \cup \{\tilde{p}_{i+1}\}$ where $\tilde{p}_{i+1} = \mathrm{root}(\varphi(p_{i+1}))$. We simultaneously show by induction on k that (a) $\varphi(\mathcal{P}_0) \overset{*}{\Rightarrow} \varphi(\mathcal{P}_k)$ and (b) if $\mathcal{P}_0 \overset{*}{\underset{I}{\Rightarrow}} \mathcal{P}_k$ then φ carries Σ_k to $\mathscr{F}_k$.

The base step for (b) follows from the condition (1). For the induction step, we divide the cases by the last inference rule applied to $\mathcal{P}_0 \overset{*}{\Rightarrow} \mathcal{P}_k$.

1. The case of $\mathcal{P}_k \underset{I}{\Rightarrow} \mathcal{P}_{k+1}$. Let $\mathcal{P}_{k+1} = \mathcal{P}_k \cup \{p_{k+1}(x_1, \ldots, x_n) \to r\}$. By the condition of the Introduction rule $x_1, \ldots, x_n$ are mutually distinct variables and $p_{k+1} \in \mathscr{X} \setminus \Sigma_k$ and $r \in \mathrm{T}(\Sigma_k, \{x_1, \ldots, x_n\})$. By the assumption (3) and the way we took $\mathscr{F}_0, \ldots, \mathscr{F}_k$, $\varphi(p_{k+1}) = f_{p_{k+1}}(\Box_1, \ldots, \Box_{\mathrm{arity}(p_{k+1})})$ with $f_{p_{k+1}} \notin \mathscr{F}_k$. Since any term homomorphism is injective on local variables, $\varphi(p_{k+1}(x_1, \ldots, x_n)) = f_p(y_1, \ldots, y_n)$ and $\mathscr{V}(\varphi(r)) \subseteq \{y_1, \ldots, y_n\}$ where $\varphi(x_i) = y_i$ with distinct local variables $y_1, \ldots, y_n$. Since φ carries Σ_k to $\mathscr{F}_k$ by the induction hypothesis and $r \in \mathrm{T}(\Sigma_k, \{x_1, \ldots, x_n\})$, we have $\varphi(r) \in \mathrm{T}(\mathscr{F}_k, \{y_1, \ldots, y_n\})$ and thus $\varphi(\mathcal{P}_k) \underset{I}{\Rightarrow} \varphi(\mathcal{P}_{k+1})$. Furthermore, since $\Sigma_{k+1} = \Sigma_k \cup \{p_{k+1}\}$, $\mathscr{F}_{k+1} = \mathscr{F}_k \cup \{f_{p_{k+1}}\}$, φ carries Σ_k to $\mathscr{F}_k$, and $\varphi(p_{k+1}) = f_{p_{k+1}}(\Box_1, \ldots \Box_{\mathrm{arity}(p_{k+1})}) \in \mathrm{T}(\mathscr{F}_k \cup \mathscr{H}_{\mathrm{arity}(p_{k+1})})$, φ carries Σ_{k+1} to $\mathscr{F}_{k+1}$.
2. The case $\mathcal{P}_k \underset{A}{\Rightarrow} \mathcal{P}_{k+1}$. Then $\mathcal{P}_{k+1} = \mathcal{P}_k \cup \{l \to r\}$ with $l \overset{*}{\leftrightarrow}_{\mathcal{P}_k \cup \mathcal{H}} r$. From Proposition 1, we have $\varphi(l) \overset{*}{\leftrightarrow}_{\varphi(\mathcal{P}_k) \cup \varphi(\mathcal{H})} \varphi(r)$. Thus $\varphi(\mathcal{P}_k) \underset{A}{\Rightarrow} \varphi(\mathcal{P}_{k+1})$.
3. The case $\mathcal{P}_k \underset{E}{\Rightarrow} \mathcal{P}_{k+1}$. Clearly, $\varphi(\mathcal{P}_k) \underset{E}{\Rightarrow} \varphi(\mathcal{P}_{k+1})$.

□

The conditions of suitable transformations are summarized in the following definition.

Definition 4: Let $\langle \mathcal{P}, \mathcal{P}', \mathcal{H} \rangle$ be a correct template where $\mathcal{P}$ and $\mathcal{P}'$ are TRS patterns over Σ and Σ', respectively. A TRS $\mathcal{R}$ over $\mathscr{G}$ *is transformed to* a TRS $\mathcal{R}'$ over $\mathscr{G}'$ by the correct template $\langle \mathcal{P}, \mathcal{P}', \mathcal{H} \rangle$ if there exist a term homomorphism φ and a TRS $\mathcal{R}_{com}$ over $\mathscr{G}$ such that:

1. $\mathcal{R} = \varphi(\mathcal{P}) \cup \mathcal{R}_{com}$,
2. φ carries Σ to $\mathscr{G}$,
3. $\mathcal{R}' = \varphi_{out}(\mathcal{P}') \cup \mathcal{R}_{com}$, where $\varphi_{out} = \varphi \cup \{p \mapsto f_p(\Box_1, \ldots, \Box_{\mathrm{arity}(p)}) \mid p \in \mathscr{X} \setminus \Sigma\}$ where each f_p is a fresh function symbol, and
4. $\mathcal{R}, \mathscr{G} \vdash_{ind} \varphi(\mathcal{H})$.

Note that although $\mathcal{R}'$ can be obtained from the templates and term homomorphism φ, our definition of transformation by correct templates also imposes the additional conditions 2 and 4 which are used to guarantee the correctness of the transformation [†].

The next theorem gives a sufficient condition to guarantee the correctness of TRS transformations by correct templates.

Theorem 1: Let $\mathscr{F}_c \subseteq \mathscr{G}, \mathscr{G}' \subseteq \mathscr{F}$. If a left-linear CS $\mathcal{R}$ over $\mathscr{G}$ is transformed to a TRS $\mathcal{R}'$ over $\mathscr{G}'$ by a correct template $\langle \mathcal{P}, \mathcal{P}', \mathcal{H} \rangle$, then $\mathrm{CR}(\mathcal{R}) \wedge \mathrm{SC}(\mathcal{R}, \mathscr{G}) \wedge \mathrm{SC}(\mathcal{R}', \mathscr{G}')$ implies $\mathcal{R} \simeq_{\mathscr{G} \cap \mathscr{G}'} \mathcal{R}'$.

Proof. Suppose that a left-linear CS $\mathcal{R}$ over $\mathscr{G}$ is transformed to a TRS $\mathcal{R}'$ over $\mathscr{G}'$ by a correct template $\langle \mathcal{P}, \mathcal{P}', \mathcal{H} \rangle$. Then there exist a term homomorphism φ and a TRS $\mathcal{R}_{com}$ over $\mathscr{G}$ that satisfies conditions of Definition 4. By the condition 3, we have $\varphi(\mathcal{P}) = \varphi_{out}(\mathcal{P})$. Furthermore, since $\mathcal{R}_{com}$ is over $\mathscr{G}$, we have $\mathcal{R} = \varphi(\mathcal{P} \cup \mathcal{R}_{com})$ and $\mathcal{R}' = \varphi(\mathcal{P}' \cup \mathcal{R}_{com})$. Then, by conditions 2, 3, 4, it follows from Lemma 1 that there is an equivalent transformation sequence from $\mathcal{R}$ to $\mathcal{R}'$ under $\varphi_{out}(\mathcal{H})$ $(= \varphi(\mathcal{H}))$. Then the claim follows from Proposition 2. □

Below we present several correct templates for tupling transformations.

Example 1: Let $\langle \mathcal{P}, \mathcal{P}', \mathcal{H} \rangle$ be a template where

$$\mathcal{P} \begin{cases} \mathsf{p}(\mathsf{a}) & \to \mathsf{b} \\ \mathsf{p}(\mathsf{c}(x,y)) & \to \mathsf{h}(\mathsf{e}(x, \mathsf{c}(x,y)), \mathsf{p}(y), \mathsf{q}(y))) \\ \mathsf{q}(\mathsf{a}) & \to \mathsf{d} \\ \mathsf{q}(\mathsf{c}(x,y)) & \to \mathsf{k}(\mathsf{e}(x, \mathsf{c}(x,y)), \mathsf{p}(y), \mathsf{q}(y)) \\ \pi_1(\langle x, y \rangle) & \to x \\ \pi_2(\langle x, y \rangle) & \to y \end{cases}$$

$$\mathcal{P}' \begin{cases} \mathsf{p}(x) & \to \pi_1(\mathsf{r}(x)) \\ \mathsf{q}(x) & \to \pi_2(\mathsf{r}(x)) \\ \mathsf{r}(\mathsf{a}) & \to \langle \mathsf{b}, \mathsf{d} \rangle \\ \mathsf{r}(\mathsf{c}(x,y)) & \to \mathsf{r1}(\mathsf{e}(x, \mathsf{c}(x,y)), \mathsf{r}(y)) \\ \mathsf{r1}(x,y) & \to \langle \mathsf{h}(x, \pi_1(y), \pi_2(y)), \mathsf{k}(x, \pi_1(y), \pi_2(y)) \rangle \\ \pi_1(\langle x, y \rangle) & \to x \\ \pi_2(\langle x, y \rangle) & \to y \end{cases}$$

$$\mathcal{H} = \emptyset.$$

Here, π_1, π_2, and $\langle \cdot \rangle$ are function symbols. We now show that the template $\langle \mathcal{P}, \mathcal{P}', \mathcal{H} \rangle$ is a correct template.

1. Let $\mathcal{P}_0 = \mathcal{P}$.
2. Let $\mathcal{P}_1 = \mathcal{P}_0 \cup \{\mathsf{r}(x) \to \langle \mathsf{p}(x), \mathsf{q}(x) \rangle\}$. Here, r is a fresh pattern variable. Thus, $\mathcal{P}_0 \Rightarrow \mathcal{P}_1$ by the Introduction rule.
3. Let $\mathcal{P}_2 = \mathcal{P}_1 \cup \{ \mathsf{r1}(x,y) \to \langle \mathsf{h}(x, \pi_1(y), \pi_2(y)), \mathsf{k}(x, \pi_1(y), \pi_2(y)) \rangle \}$. Here, $\mathsf{r1}$ is a fresh pattern variable. Thus, $\mathcal{P}_1 \Rightarrow \mathcal{P}_2$ by the Introduction rule.
4. Let $\mathcal{P}_3 = \mathcal{P}_2 \cup \{\mathsf{r}(\mathsf{a}) \to \langle \mathsf{b}, \mathsf{d} \rangle\}$. Here, we have $\mathsf{r}(\mathsf{a}) \to_{\mathcal{P}_2} \langle \mathsf{p}(\mathsf{a}), \mathsf{q}(\mathsf{a}) \rangle \overset{*}{\to}_{\mathcal{P}_2} \langle \mathsf{b}, \mathsf{d} \rangle$. Thus, $\mathcal{P}_2 \Rightarrow \mathcal{P}_3$ by the Addition rule.
5. Let $\mathcal{P}_4 = \mathcal{P}_3 \cup \{\mathsf{r}(\mathsf{c}(x,y)) \to \mathsf{r1}(\mathsf{e}(x, \mathsf{c}(x,y)), \mathsf{r}(y))\}$. Here, we have $\mathsf{r}(\mathsf{c}(x,y)) \to_{\mathcal{P}_3} \langle \mathsf{p}(\mathsf{c}(x,y)), \mathsf{q}(\mathsf{c}(x,y)) \rangle$

[†] We regard the verification of these conditions as a part of verification of correctness of the program transformation by templates.

$$
\begin{aligned}
&\overset{*}{\to}_{\mathcal{P}_3} \langle \mathsf{h}(\mathsf{e}(x, \mathsf{c}(x, y)), \mathsf{p}(y), \mathsf{q}(y)),\\
&\qquad \mathsf{k}(\mathsf{e}(x, \mathsf{c}(x, y)), \mathsf{p}(y), \mathsf{q}(y))\rangle\\
&\overset{*}{\leftarrow}_{\mathcal{P}_3}\\
&\quad \langle \mathsf{h}(\mathsf{e}(x, \mathsf{c}(x, y)), \pi_1(\langle \mathsf{p}(y), \mathsf{q}(y)\rangle), \pi_2(\langle \mathsf{p}(y), \mathsf{q}(y)\rangle)),\\
&\quad\ \mathsf{k}(\mathsf{e}(x, \mathsf{c}(x, y)), \pi_1(\langle \mathsf{p}(y), \mathsf{q}(y)\rangle), \pi_2(\langle \mathsf{p}(y), \mathsf{q}(y)\rangle))\rangle\\
&\overset{*}{\leftarrow}_{\mathcal{P}_3} \langle \mathsf{h}(\mathsf{e}(x, \mathsf{c}(x, y)), \pi_1(\mathsf{r}(y)), \pi_2(\mathsf{r}(y))),\\
&\qquad \mathsf{k}(\mathsf{e}(x, \mathsf{c}(x, y)), \pi_1(\mathsf{r}(y)), \pi_2(\mathsf{r}(y)))\rangle\\
&\leftarrow_{\mathcal{P}_3} \mathsf{r1}(\mathsf{e}(x, \mathsf{c}(x, y)), \mathsf{r}(y)).
\end{aligned}
$$

Thus, $\mathcal{P}_3 \Rightarrow \mathcal{P}_4$ by the Addition rule.

6. Let $\mathcal{P}_5 = \mathcal{P}_4 \cup \{\mathsf{p}(x) \to \pi_1(\mathsf{r}(x))\}$. Here, we have $\mathsf{p}(x) \leftarrow_{\mathcal{P}_4} \pi_1(\langle \mathsf{p}(x), \mathsf{q}(x)\rangle) \leftarrow_{\mathcal{P}_4} \pi_1(\mathsf{r}(x))$. Thus, $\mathcal{P}_4 \Rightarrow \mathcal{P}_5$ by the Addition rule.
7. Let $\mathcal{P}_6 = \mathcal{P}_5 \cup \{\mathsf{q}(x) \to \pi_2(\mathsf{r}(x))\}$. Here, we have $\mathsf{q}(x) \leftarrow_{\mathcal{P}_5} \pi_2(\langle \mathsf{p}(x), \mathsf{q}(x)\rangle) \leftarrow_{\mathcal{P}_5} \pi_2(\mathsf{r}(x))$. Thus, $\mathcal{P}_5 \Rightarrow \mathcal{P}_6$ by the Addition rule.
8. Finally, applying the Elimination rule five times to $\mathcal{P}_6$, we obtain $\mathcal{P}'$.

Thus $\mathcal{P} \overset{*}{\underset{I}{\Rightarrow}} \cdot \overset{*}{\underset{A}{\Rightarrow}} \cdot \overset{*}{\underset{E}{\Rightarrow}} \mathcal{P}'$ under $\mathcal{H}$ and hence $\langle \mathcal{P}, \mathcal{P}', \mathcal{H}\rangle$ is a correct template.

The transformation from TRS $\mathcal{R}_{\mathsf{factlist}} \cup \mathcal{R}_\pi$ to $\mathcal{R}'_{\mathsf{factlist}}$ in Sect. 3 is obtained by this template as follows. By matching the source $\mathcal{P}$ and the TRS $(\mathcal{R}_{\mathsf{factlist}} \cup \mathcal{R}_\pi) \setminus \mathcal{R}_{\mathsf{com}}$, we obtain the term homomorphism $\varphi =$

$$
\left\{
\begin{array}{ll}
\mathsf{p} \mapsto \mathsf{factlist}(\Box_1) & \mathsf{a} \mapsto \mathsf{0}\\
\mathsf{b} \mapsto [\,] & \mathsf{c} \mapsto \mathsf{s}(\Box_2)\\
\mathsf{h} \mapsto \Box_3{:}\Box_2 & \mathsf{e} \mapsto \Box_2\\
\mathsf{q} \mapsto \mathsf{fact}(\Box_1) & \mathsf{d} \mapsto \mathsf{s}(\mathsf{0})\\
\mathsf{k} \mapsto \times(\Box_1, \Box_3) &
\end{array}
\right\}
$$

where $\mathcal{R}_{\mathsf{com}} =$

$$
\left\{
\begin{array}{lcl}
\times(\mathsf{0}, y) & \to & \mathsf{0}\\
\times(\mathsf{s}(x), y) & \to & +(y, \times(x, y))\\
+(\mathsf{0}, x) & \to & x\\
+(\mathsf{s}(x), y) & \to & \mathsf{s}(+(x, y)).
\end{array}
\right.
$$

It is checked automatically that the term homomorphism φ carries $\{\mathsf{p}, \mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{h}, \mathsf{e}, \mathsf{q}, \mathsf{d}, \mathsf{k}, \pi_1, \pi_2, \langle\cdot\rangle\}$ to $\{\mathsf{factlist}, \mathsf{fact}, \times, +, :, [\,], \mathsf{s}, \mathsf{0}, \pi_1, \pi_2, \langle\cdot\rangle\}$. We have $\mathcal{R}_{\mathsf{factlist}'} = \tilde{\varphi}(\mathcal{P}') \cup \mathcal{R}_{\mathsf{com}}$ where $\tilde{\varphi} = \varphi \circ \{\mathsf{r} \mapsto \mathsf{factpair}(\Box_1), \mathsf{r1} \mapsto \mathsf{step}(\Box_1, \Box_2)\}$. Since $\mathcal{H} = \emptyset$, $\mathcal{R}_{\mathsf{factlist}} \cup \mathcal{R}_\pi, \mathscr{G} \vdash_{ind} \varphi(\mathcal{H})$ holds trivially. Hence the TRS $\mathcal{R}_{\mathsf{factlist}} \cup \mathcal{R}_\pi$ is transformed to $\mathcal{R}'_{\mathsf{factlist}}$.

As explained in the end of Section 4, $\mathrm{CR}(\mathcal{R}_{\mathsf{factlist}} \cup \mathcal{R}_\pi)$, $\mathrm{SC}(\mathcal{R}_{\mathsf{factlist}} \cup \mathcal{R}_\pi, \mathscr{G})$ and $\mathrm{SC}(\mathcal{R}'_{\mathsf{factlist}}, \mathscr{G}')$ can be proved automatically (under the assumption of many-sorted signature). The correctness of the transformation also follows from Theorem 1, i.e. $\mathcal{R}_{\mathsf{factlist}} \cup \mathcal{R}_\pi \simeq_{\{\mathsf{factlist},\mathsf{fact},\times,+,:,[\,],\mathsf{s},\mathsf{0},\pi_1,\pi_2,\langle\cdot\rangle\}} \mathcal{R}'_{\mathsf{factlist}}$.

Example 2: A list of numbers is said to be *steep* if each element is greater than the sum of the elements that follow it [12]. The following TRS $\mathcal{R}_{\mathsf{steep}}$ specifies a program which checks whether the input list is steep. $\mathcal{R}_{\mathsf{steep}} =$

$$
\mathcal{R}_{\mathsf{com}} \cup \left\{
\begin{array}{ll}
\mathsf{steep}([\,]) & \to \mathsf{tt}\\
\mathsf{steep}(x{:}xs) & \to \mathsf{and}(\mathsf{gt}(x, \mathsf{sum}(xs)), \mathsf{steep}(xs))\\
\mathsf{sum}([\,]) & \to \mathsf{0}\\
\mathsf{sum}(x{:}ys) & \to +(x, \mathsf{sum}(ys))\\
\pi_1(\langle x, y\rangle) & \to x\\
\pi_2(\langle x, y\rangle) & \to y
\end{array}
\right.
$$

where $\mathcal{R}_{\mathsf{com}} =$

$$
\left\{
\begin{array}{llllll}
\mathsf{gt}(\mathsf{0}, \mathsf{0}) & \to \mathsf{ff} & \mathsf{gt}(\mathsf{s}(x), \mathsf{0}) & \to \mathsf{tt}\\
\mathsf{gt}(\mathsf{0}, \mathsf{s}(y)) & \to \mathsf{ff} & \mathsf{gt}(\mathsf{s}(x), \mathsf{s}(y)) & \to \mathsf{gt}(x, y)\\
\mathsf{and}(\mathsf{tt}, \mathsf{tt}) & \to \mathsf{tt} & \mathsf{and}(\mathsf{tt}, \mathsf{ff}) & \to \mathsf{ff}\\
\mathsf{and}(\mathsf{ff}, \mathsf{tt}) & \to \mathsf{ff} & \mathsf{and}(\mathsf{ff}, \mathsf{ff}) & \to \mathsf{ff}\\
+(\mathsf{0}, x) & \to x & +(\mathsf{s}(x), y) & \to \mathsf{s}(+(x, y)).
\end{array}
\right.
$$

Let $\langle \mathcal{P}, \mathcal{P}', \mathcal{H}\rangle$ be the template in Example 1. The following term homomorphism φ is a matcher between the source $\mathcal{P}$ and the TRS $\mathcal{R}_{\mathsf{steep}} \setminus \mathcal{R}_{\mathsf{com}}$, that is, $\varphi(\mathcal{P}) = \mathcal{R}_{\mathsf{steep}} \setminus \mathcal{R}_{\mathsf{com}}$.

$$
\varphi = \left\{
\begin{array}{l}
\mathsf{p} \mapsto \mathsf{steep}(\Box_1)\\
\mathsf{a} \mapsto [\,]\\
\mathsf{b} \mapsto \mathsf{tt}\\
\mathsf{c} \mapsto \Box_1{:}\Box_2\\
\mathsf{h} \mapsto \mathsf{and}(\mathsf{gt}(\Box_1, \Box_3), \Box_2)\\
\mathsf{e} \mapsto \Box_1\\
\mathsf{q} \mapsto \mathsf{sum}(\Box_1)\\
\mathsf{d} \mapsto \mathsf{0}\\
\mathsf{k} \mapsto +(\Box_1, \Box_3)
\end{array}
\right\}
$$

It is easy to check that the above term homomorphism φ carries $\{\mathsf{p}, \mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{h}, \mathsf{e}, \mathsf{q}, \mathsf{d}, \mathsf{k}, \pi_1, \pi_2, \langle\cdot\rangle\}$ to $\{\mathsf{steep}, \mathsf{sum}, \mathsf{gt}, \mathsf{and}, +, :, [\,], \mathsf{tt}, \mathsf{ff}, \mathsf{s}, \mathsf{0}, \pi_1, \pi_2, \langle\cdot\rangle\}$. Since $\mathcal{H} = \emptyset$, $\mathcal{R}_{\mathsf{steep}}, \mathscr{G} \vdash_{ind} \varphi(\mathcal{H})$ holds trivially. Let $\mathcal{R}'_{\mathsf{steep}} = \mathcal{R}_{\mathsf{com}} \cup \tilde{\varphi}(\mathcal{P}') =$

$$
\mathcal{R}_{\mathsf{com}} \cup \left\{
\begin{array}{l}
\mathsf{steep}(x) \to \pi_1(\mathsf{f_r}(x))\\
\mathsf{sum}(x) \to \pi_2(\mathsf{f_r}(x))\\
\mathsf{f_r}([\,]) \to \langle \mathsf{tt}, \mathsf{0}\rangle\\
\mathsf{f_r}(x{:}y) \to \mathsf{f_{r1}}(x, \mathsf{f_r}(y))\\
\mathsf{f_{r1}}(x, y) \to\\
\quad \langle \mathsf{and}(\mathsf{gt}(x, \pi_2(y)), \pi_1(y)), +(x, \pi_2(y))\rangle\\
\pi_1(\langle x, y\rangle) \to x\\
\pi_2(\langle x, y\rangle) \to y
\end{array}
\right.
$$

where $\tilde{\varphi} = \varphi \circ \{\mathsf{r} \mapsto \mathsf{f_r}(\Box_1), \mathsf{r1} \mapsto \mathsf{f_r1}(\Box_1)\}$. Thus the TRS $\mathcal{R}_{\mathsf{steep}}$ is transformed to the TRS $\mathcal{R}'_{\mathsf{steep}}$ by the template $\langle \mathcal{P}, \mathcal{P}', \mathcal{H}\rangle$.

The termination of the TRSs $\mathcal{R}_{\mathsf{steep}}$ and $\mathcal{R}'_{\mathsf{steep}}$ can be proved automatically using the lexicographic path order; from this, $\mathrm{CR}(\mathcal{R}_{\mathsf{steep}})$, $\mathrm{SC}(\mathcal{R}_{\mathsf{steep}}, \mathscr{G})$ and $\mathrm{SC}(\mathcal{R}'_{\mathsf{steep}}, \mathscr{G}')$ are checked by computing the critical pairs and the complement substitution algorithm (under the assumption of many-sorted signature). Therefore, the correctness of the transformation is guaranteed by Theorem 1, that is, $\mathcal{R}_{\mathsf{steep}} \simeq_{\{\mathsf{steep},\mathsf{sum},\mathsf{gt},\mathsf{and},+,:,[\,],\mathsf{tt},\mathsf{ff},\mathsf{s},\mathsf{0},\pi_1,\pi_2,\langle\cdot\rangle\}} \mathcal{R}'_{\mathsf{steep}}$.

Example 3: Let $\langle \tilde{\mathcal{P}}, \tilde{\mathcal{P}}', \tilde{\mathcal{H}}\rangle$ be a template where

$$\tilde{\mathcal{P}}\left\{\begin{array}{lcl}\mathsf{p(a)} & \to & \mathsf{c}\\ \mathsf{p(b(a))} & \to & \mathsf{d}\\ \mathsf{p(b(b}(x))) & \to & \mathsf{q(p(b}(x)), \mathsf{p}(x))\\ \pi_1(\langle x, y\rangle) & \to & x\\ \pi_2(\langle x, y\rangle) & \to & y\end{array}\right.$$

$$\tilde{\mathcal{P}}'\left\{\begin{array}{lcl}\mathsf{p(a)} & \to & \mathsf{c}\\ \mathsf{p(b(a))} & \to & \mathsf{d}\\ \mathsf{p(b(b}(x))) & \to & \pi_1(\mathsf{r1(r}(x)))\\ \mathsf{r(a)} & \to & \langle \mathsf{d}, \mathsf{c}\rangle\\ \mathsf{r(b}(x)) & \to & \mathsf{r1(r}(x))\\ \mathsf{r1}(x) & \to & \langle \mathsf{q}(\pi_1(x), \pi_2(x)), \pi_1(x)\rangle\\ \pi_1(\langle x, y\rangle) & \to & x\\ \pi_2(\langle x, y\rangle) & \to & y\end{array}\right.$$

$\tilde{\mathcal{H}} = \emptyset$.

We now show that the template $\langle \tilde{\mathcal{P}}, \tilde{\mathcal{P}}', \tilde{\mathcal{H}}\rangle$ is a correct template.

1. Let $\mathcal{P}_0 = \tilde{\mathcal{P}}$.
2. Let $\mathcal{P}_1 = \mathcal{P}_0 \cup \{\mathsf{r}(x) \to \langle \mathsf{p(b}(x)), \mathsf{p}(x)\rangle\}$. Here, r is a fresh pattern variable. Thus, $\mathcal{P}_0 \Rightarrow \mathcal{P}_1$ by the Introduction rule.
3. Let $\mathcal{P}_2 = \mathcal{P}_1 \cup \{\mathsf{r1}(x) \to \langle \mathsf{q}(\pi_1(x), \pi_2(x)), \pi_1(x)\rangle\}$. Here, r1 is a fresh pattern variable. Thus, $\mathcal{P}_1 \Rightarrow \mathcal{P}_2$ by the Introduction rule.
4. Let $\mathcal{P}_3 = \mathcal{P}_2 \cup \{\mathsf{r(a)} \to \langle \mathsf{d}, \mathsf{c}\rangle\}$. Here, we have $\mathsf{r(a)} \to_{\mathcal{P}_2} \langle \mathsf{p(b(a))}, \mathsf{p(a)}\rangle \overset{*}{\to}_{\mathcal{P}_2} \langle \mathsf{d}, \mathsf{c}\rangle$. Thus, $\mathcal{P}_2 \Rightarrow \mathcal{P}_3$ by the Addition rule.
5. Let $\mathcal{P}_4 = \mathcal{P}_3 \cup \{\mathsf{r(b}(x)) \to \mathsf{r1(r}(x))\}$. Here, we have

$$\begin{array}{lll}\mathsf{r(b}(x)) & \to_{\mathcal{P}_3} & \langle \mathsf{p(b(b}(x))), \mathsf{p(b}(x))\rangle\\ & \to_{\mathcal{P}_3} & \langle \mathsf{q(p(b}(x)), \mathsf{p}(x)), \mathsf{p(b}(x))\rangle\\ & \overset{*}{\leftarrow}_{\mathcal{P}_3} & \langle \mathsf{q}(\pi_1(\mathsf{r}(x)), \pi_2(\mathsf{r}(x))), \pi_1(\mathsf{r}(x))\rangle\\ & \leftarrow_{\mathcal{P}_3} & \mathsf{r1(r}(x)).\end{array}$$

   Thus, $\mathcal{P}_3 \Rightarrow \mathcal{P}_4$ by the Addition rule.
6. Let $\mathcal{P}_5 = \mathcal{P}_4 \cup \{\mathsf{p(b(b}(x))) \to \pi_1(\mathsf{r1(r}(x)))\}$. Here, we have

$$\begin{array}{lll}\mathsf{p(b(b}(x))) & \leftarrow_{\mathcal{P}_4} & \pi_1(\langle \mathsf{p(b(b}(x))), \mathsf{p(b}(x))\rangle)\\ & \leftarrow_{\mathcal{P}_4} & \pi_1(\mathsf{r(b}(x)))\\ & \to_{\mathcal{P}_4} & \pi_1(\mathsf{r1(r}(x))).\end{array}$$

   Thus, $\mathcal{P}_4 \Rightarrow \mathcal{P}_5$ by the Addition rule.
7. Finally, applying the Elimination rule twice to $\mathcal{P}_5$, we obtain $\tilde{\mathcal{P}}'$.

Thus $\tilde{\mathcal{P}} \underset{I}{\overset{*}{\Rightarrow}} \cdot \underset{A}{\overset{*}{\Rightarrow}} \cdot \underset{E}{\overset{*}{\Rightarrow}} \tilde{\mathcal{P}}'$ under $\tilde{\mathcal{H}}$ and hence $\langle \tilde{\mathcal{P}}, \tilde{\mathcal{P}}', \tilde{\mathcal{H}}\rangle$ is a correct template.

The transformation from TRS $\mathcal{R}_{\mathsf{fib}} \cup \mathcal{R}_\pi$ to $\mathcal{R}'_{\mathsf{fib}}$ in Sect. 3 is obtained by this template as follows. By matching the source $\mathcal{P}$ and the TRS $(\mathcal{R}_{\mathsf{fib}} \cup \mathcal{R}_\pi) \setminus \tilde{\mathcal{R}}_{\mathsf{com}}$, we obtain the term homomorphism $\varphi =$

$$\left\{\begin{array}{lcl}\mathsf{p} & \mapsto & \mathsf{fib}(\Box_1)\\ \mathsf{a} & \mapsto & \mathsf{0}\\ \mathsf{c} & \mapsto & \mathsf{s(0)}\\ \mathsf{b} & \mapsto & \mathsf{s}(\Box_1)\\ \mathsf{d} & \mapsto & \mathsf{s(0)}\\ \mathsf{q} & \mapsto & +(\Box_1, \Box_2)\end{array}\right\}$$

where

$$\tilde{\mathcal{R}}_{\mathsf{com}} = \left\{\begin{array}{lcl}+(\mathsf{0}, x) & \to & x\\ +(\mathsf{s}(x), y) & \to & \mathsf{s}(+(x, y)).\end{array}\right.$$

It is easily checked that φ carries $\{\mathsf{p}, \mathsf{a}, \mathsf{c}, \mathsf{b}, \mathsf{d}, \mathsf{q}, \pi_1, \pi_2, \langle\cdot\rangle\}$ to $\{\mathsf{fib}, +, \mathsf{s}, \mathsf{0}, \pi_1, \pi_2, \langle\cdot\rangle\}$. Since $\tilde{\mathcal{H}} = \emptyset$, $\mathcal{R}_{\mathsf{fib}} \cup \mathcal{R}_\pi, \mathcal{G} \vdash_{ind} \varphi(\tilde{\mathcal{H}})$ holds trivially. Since $\mathcal{R}'_{\mathsf{fib}} = \tilde{\mathcal{R}}_{\mathsf{com}} \cup \tilde{\varphi}(\mathcal{P}')$ where $\tilde{\varphi} = \varphi \circ \{\mathsf{r} \mapsto \mathsf{fibpair}(\Box_1), \mathsf{r1} \mapsto \mathsf{step}(\Box_1)\}$, the TRS $\mathcal{R}_{\mathsf{fib}} \cup \mathcal{R}_\pi$ is transformed to the TRS $\mathcal{R}'_{\mathsf{fib}}$.

The termination of the TRSs $\mathcal{R}_{\mathsf{fib}} \cup \mathcal{R}_\pi$ and $\mathcal{R}'_{\mathsf{fib}}$ can be shown automatically by lexicographic path order; from this, $\mathrm{CR}(\mathcal{R}_{\mathsf{fib}} \cup \mathcal{R}_\pi)$, $\mathrm{SC}(\mathcal{R}_{\mathsf{fib}} \cup \mathcal{R}_\pi, \mathcal{G})$ and $\mathrm{SC}(\mathcal{R}'_{\mathsf{fib}}, \mathcal{G}')$ are checked by computing the critical pairs and the complement substitution algorithm (under the assumption of many-sorted signature). Therefore, the correctness of the transformation is guaranteed by Theorem 1, that is, $\mathcal{R}_{\mathsf{fib}} \cup \mathcal{R}_\pi \simeq_{\{\mathsf{fib},+,\mathsf{s},\mathsf{0},\pi_1,\pi_2,\langle\cdot\rangle\}} \mathcal{R}'_{\mathsf{fib}}$.

6. Comparison with the Previous Framework

As mentioned earlier, in our previous framework, we guarantee the successful lift up of abstract equivalent transformation sequence by the notions of CS homomorphism and developed templates. In this section, we compare our new framework with the previous one given in [6], [7].

Definition 5 ([6], [7]): Suppose $\mathscr{P}$ is partitioned into sets $\mathscr{P}_\mathrm{d}$, $\mathscr{P}_\mathrm{c}$ of defined pattern symbols and constructor pattern symbols. A term homomorphism φ is said to be a *CS homomorphism* if (1) for any $p \in \mathscr{P}_\mathrm{d}$ of arity n, $\varphi(p) = f(\Box_{i_1}, \ldots, \Box_{i_n})$ for some $f \in \mathscr{F}_\mathrm{d}$ and mutually distinct indexes $i_1, \ldots, i_n$; and (2) for any $p, q \in \mathscr{P}_\mathrm{d}$, $p \neq q$ implies $\mathrm{root}(\varphi(p)) \neq \mathrm{root}(\varphi(q))$. A template is *developed* if there is an abstract equivalent transformation sequence such that $p(x_1, \ldots, x_n) \to r$ in the Introduction rule is restricted as $p \in \mathscr{P}_\mathrm{d} \setminus \Sigma_k$ and $r \in \mathrm{T}(\Sigma_k \cap \mathscr{P}_\mathrm{d}, \mathscr{V})$.

It is easy to see that $\mathcal{P} \underset{I}{\Rightarrow} \mathcal{P}'$ under the restricted Introduction rule implies $\varphi(\mathcal{P}) \underset{I}{\Rightarrow} \varphi(\mathcal{P}')$ for any CS homomorphism φ [6], [7].

As demonstrated in Examples 1 and 3, templates for tupling transformations need to introduce rewrite rules $\mathsf{r}(x) \to \langle s_1, \ldots, s_n\rangle$ and $\mathsf{r1}(x) \to \langle s'_1, \ldots, s'_m\rangle$ using a tuple symbol $\langle\cdot\rangle$ by the Introduction rule. However, since $\langle\cdot\rangle \in \mathscr{F}_\mathrm{c}$, one can not introduce such a rewrite rule by the condition of abstract equivalent transformation sequence for developed templates that rhs of the introduced rule should be contained in $\mathrm{T}(\Sigma_k \cap \mathscr{P}_\mathrm{d}, \mathscr{V})$. Hence templates for tupling transformations can not be expressed by developed templates. This is why the previous framework of [5]–[8] fails to handle tupling transformations.

In the previous framework, templates are constructed on $\mathrm{T}(\mathscr{P}, \mathscr{V})$ and thus function symbols are excluded. Then one may wonder why a pattern variable $q \in \mathscr{P}_\mathrm{c}$ had not been allowed in the Introduction rule for the developed templates. In general, if one allows a pattern variable $q \in \mathscr{P}_\mathrm{c}$ occur in the rhs r of the rewrite rule introduced by the Introduction

rule, then, for a CS homomorphism φ, $\mathcal{P} \underset{I}{\Rightarrow} \mathcal{P}'$ may not imply $\varphi(\mathcal{P}) \underset{I}{\Rightarrow} \varphi(\mathcal{P}')$. For example, let $\mathsf{p}(x, y) \to \mathsf{q}(x, \mathsf{r}(x, y))$ be a rewrite rule in the input template and $\varphi = \{\mathsf{p} \mapsto \mathsf{f}(\Box_1, \Box_2), \mathsf{q} \mapsto \mathsf{g}(\Box_1), \mathsf{r} \mapsto \mathsf{h}(\Box_1, \Box_2)\}$ be a matcher. Then one can not guarantee $\mathsf{h} \in \mathscr{F}_k$ because $\mathsf{h} \notin \mathscr{F}(\varphi(\mathsf{q}(x, \mathsf{r}(x, y))))$. One can not exclude the case of $\mathsf{r} \in \Sigma_k$ but $\mathscr{F}(\varphi(\mathsf{r})) \not\subseteq \mathscr{F}_k$. In our new criterion, such a case is forbidden by introducing the notion of carrying of signature.

7. Conclusion

We extended a framework of program transformation by templates based on term rewriting. We introduced a notion of correct templates by which a wider variety of transformations by template, including tupling transformations, can be handled. We proved a criterion to guarantee the correctness of program transformations by these templates. We gave several correct templates for tupling transformations and demonstrated some examples of tupling transformations *by templates*.

RAPT [†][5] is an implementation of program transformation by templates based on term rewriting. For a given TRS $\mathcal{R}$ and a correct template, it automatically verifies the correctness of transformation of $\mathcal{R}$ by that template, and outputs the result of transformation. We have implemented the new version of RAPT based on the framework presented in this paper. For all examples in this paper, program transformations and their verification have been operated in RAPT successfully. To complete the each program transformation and verification of Examples 2, 3, 4, it took 820, 620 and 40 msec, respectively, on a workstation with UltraSPARC IIIi 1.34 GHz CPU. Furthermore, it is confirmed experimentally that the new version is also capable of all examples prepared for the old version.

Another implementation of program transformation using templates is the MAG system [13], [20]. MAG supports transformations that include modifications of expressions and matching with the help of hypothesis; its target also includes higher-order programs. In contrast, the target of RAPT is limited to first-order term rewriting systems but it cooperates with automated theorem-proving techniques of term rewriting to verify the correctness of transformations.

In general, constructing a suitable template for a particular program transformation is operated manually and it is often not easy to construct a correct template that is widely applicable to many programs. To improve this situation, we are working toward the development of a method for constructing possible candidates of template automatically [8].

Another possible future extension is to extend the framework to deal with *higher-order* functions. We expect that the equivalent transformation technique can be extended to higher-order TRSs without much difficulties. However, an integration of automated verification methods for properties such as sufficient completeness, and an incorporation of higher-order inductive theorem proving [1], [17] seem yet not clear. The extension toward this direction remains as a future work.

[†]Available from `http://www.jaist.ac.jp/~chiba/RAPT/`

References

[1] T. Aoto, Y. Yamada, and Y. Toyama, "Inductive theorems for higher-order rewriting," Proc. RTA2004, vol.3091 of LNCS, pp.269–284, Springer-Verlag, 2004.
[2] F. Baader and T. Nipkow, "Term rewriting and all that," Cambridge University Press, 1998.
[3] R.S. Bird, "Using circular programs to eliminate multiple traversals of data," Acta Informatica, vol.21, pp.239–250, 1984.
[4] R.M. Burstall and J. Darlington, A transformation system for developing recursive programs, J. ACM, vol.24, no.1, pp.44–67, 1977.
[5] Y. Chiba and T. Aoto, "RAPT: A program transformation system based on term rewriting," Proc. RTA 2006, vol.4098 of LNCS, pp.267–276, Springer-Verlag, 2006.
[6] Y. Chiba, T. Aoto, and Y. Toyama, "Program transformation by templates based on term rewriting," Proc. PPDP 2005, pp.59–69, ACM Press, 2005.
[7] Y. Chiba, T. Aoto, and Y. Toyama, "Program transformation by templates: A rewriting framework," IPSJ Trans. Programming, 47 (SIG 16 (PRO 31)), pp.52–65, 2006.
[8] Y. Chiba, T. Aoto, and Y. Toyama, "Automatic construction of program transformation templates," IPSJ Trans. Programming, 49 (SIG 1 (PRO 35)), pp.14–27, 2008.
[9] W.N. Chin, "Towards an automated tupling strategy," Proc. PEPM'93, pp.119–132, ACM Press, 1993.
[10] W.N. Chin and Z. Hu, "Towards a modular program derivation via fusion and tupling," Proc. GPCE 2002, vol.2487 of LNCS, pp.140–155, Springer-Verlag, 2002.
[11] R. Curien, Z. Qian, and H. Shi, "Efficient second-order matching," Proc. RTA'96, vol.1103 of LNCS, pp.317–331, Springer-Verlag, 1996.
[12] O. de Moor and G. Sittampalam, "Generic program transformation," Proc. 3rd International Summer School on Advanced Functional Programming, vol.1608 of LNCS, pp.116–149, Springer-Verlag, 1999.
[13] O. de Moor and G. Sittampalam, "Higher-order matching for program transformation," TCS, 269, pp.135–162, 2001.
[14] A. Gill, J. Launchbury, and S. Peyton-Jones, "A short cut to deforestation," Proc. FPCA'93, pp.223–232, ACM Press, 1993.
[15] Z. Hu, H. Iwasaki, M. Takeichi, and A. Takano, "Tupling calculation eliminates multiple data traversals," Proc. ICFP'97, pp.164–175, ACM Press, 1997.
[16] G. Huet and B. Lang, "Proving and applying program transformations expressed with second order patterns," Acta Informatica, vol.11, pp.31–55, 1978.
[17] K. Kusakari, M. Sakai, and T. Sakabe, "Primitive inductive theorems bridge implicit induction methods and inductive theorems in higher-order rewriting," IEICE Trans. Inf. & Syst., vol.E88-D, no.12, pp.2715–2726, Dec. 2005.
[18] A. Lazrek, P. Lescanne, and J.J. Thiel, "Tools for proving inductive equalities, relative completeness, and ω-completeness," Information and Computation, vol.84, pp.47–70, 1990.
[19] U.S. Reddy, "Term rewriting induction," Proc. 10th International Conference on Automated Deduction, vol.449 of LNAI, pp.162–177, 1990.
[20] G. Sittampalam, Higher-order matching for program transformation, PhD Thesis, Magdalen College, 2001.
[21] J.J. Thiel, "Stop loosing sleep over incomplete data type specifications," Proc. POPL'84, pp.76–82, 1984.
[22] Y. Toyama, "How to prove equivalence of term rewriting systems without induction," TCS, vol.90, pp.369–390, 1991.
[23] P. Wadler, "Deforestation: Transforming programs to eliminate trees," TCS, vol.73, pp.231–248, 1990.

[24] T. Yokoyama, Z. Hu, and M. Takeichi, "Deterministic second-order patterns," IPL, vol.89, no.6, pp.309–314, 2004.

Yuki Chiba received his M.S. and Ph.D. from Tohoku University in 2005 and 2008. He has been in Japan Advanced Institute for Science and Technology (JAIST) as an assistant professor. His research interests include term rewriting and program transformation. He is a member of IPSJ and JSSST.

Takahito Aoto received his M.S. and Ph.D. from Japan Advanced Institute for Science and Technology (JAIST). He was at JAIST from 1997 to 1998 as an associate, at Gunma University from 1998 to 2002 as an assistant professor, and at Tohoku University from 2003 to 2004 as a lecturer. He has been in Tohoku University from 2004 as an associate professor. His current research interests include term rewriting, automated theorem proving, and foundation of software. He is a member of IPSJ, JSSST, EATCS, and ACM.

Yoshihito Toyama was born in 1952. He received his B.E. from Niigata University in 1975, and his M.E. and Ph.D. from Tohoku University in 1977 and 1990. He worked as a Research Scientist at NTT Laboratories from 1977 to 1993, and as a Professor at the Japan Advanced Institute of Science and Technology (JAIST) from 1993 to 2000. Since April 2000, he has been a professor of the Research Institute of Electrical Communication (RIEC), Tohoku University. His research interests include term rewriting systems, program theory, and automated theorem proving. He is a member of IPSJ, JSSST, ACM, and EATCS.