Regular Paper

Program Transformation by Templates: A Rewriting Framework

Yuki Chiba,† Takahito Aoto† and Yoshihito Toyama†

We propose a framework in this paper for transforming programs with templates based on term rewriting. The programs are given by term rewriting systems. We discuss how to validate the correctness of program transformation within our framework. We introduce a notion of developed templates and a simple method of constructing such templates without explicit use of induction. We then show that in any transformation of programs using the developed templates, their correctness can be verified automatically. The correctness of program transformation within our framework is discussed based on operational semantics. We also present some examples of program transformations in our framework.

1. Introduction

Automatically transforming given programs to optimize efficiency is one of the most fascinating techniques for programming languages [12),13)]. Several techniques for transforming functional programming languages have been developed [2),11),19)]. Huet and Lang [11)] presented a framework of automated program transformation in which programs are transformed according to a given program transformation template, where the template consists of program schemas for input and output programs, and a set of equations the input (and output) programs must validate to guarantee the correctness of transformation. The programs and program schemas in their framework are given by second-order simply-typed lambda terms. They gave a second-order matching algorithm to verify whether a template could be applied to an input program. They also showed how to validate the correctness of program transformation using the denotational semantics.

After Huet and Lang's pioneering work, Curien, et al. [7)] provided an improved matching algorithm using top-down matching method. Yokoyama, et al. [20)] presented sufficient conditions to have at most one solution and a deterministic algorithm to find such a solution. Recently, de Moor and Sittampalam [9)] presented a matching algorithm that could also be applied to third-order matching problems. The programs in all of these algorithms are represented by lambda terms and higher-order substitutions are achieved by the β-reduction of lambda calculus. However, in contrast to this successive work on matching algorithms, the formal verification component of the correctness of transformation has been neglected within the framework of program transformation using templates. Thus, the verification of the correctness of transformation in this framework still depends on Huet and Lang's original technique based on denotational semantics.

We propose a framework of program transformation using templates in this paper based on term rewriting. In contrast to the existing work previously mentioned, programs and program schemas are given by term rewriting systems (TRSs for short) and TRS patterns. Thus, there is little difficulty in discussing the correctness of transformation based on operational semantics.

We introduce the notion of term homomorphisms to describe how a TRS pattern matches a concrete TRS. A key part of our procedure of TRS transformation using templates——the TRS pattern matching problem——is solved using the term pattern matching algorithm [4)].

To guarantee the correctness of transformation, we also introduce the notion of developed templates and a simple method of constructing such templates without explicit use of induction. We then show that in any transformations of programs using the developed templates, the correctness of transformation can be verified automatically.

The rest of the paper is organized as follows. In Section 2, we recall basic notions in term rewriting that will be used throughout this paper. In Section 3, we present some motivating

† Research Institute of Electrical Communication, Tohoku University

☆ Part of this paper was published as a preliminary version in the proceedings of PPDP'05 [4)].

examples of TRS transformations using templates, and introduce our framework. We discuss how the correctness of TRS transformation is validated using templates in Section 4. We first describe a simple technique to prove the equality of two TRSs using a manual transformation from one to the other. We then introduce notions of CS homomorphisms and developed templates to conduct such a manual transformation at the template level. In Section 5, we briefly explain RAPT, which implements our framework. Various other examples executed by RAPT are given in Section 6. We conclude with the results in Section 7.

2. Preliminaries

This section introduces notions of term rewriting systems [1),16)].

Let $\mathscr{F}$ and $\mathscr{V}$ be a set of *function symbols* and *variables*, respectively. We assume that these sets are mutually disjoint. Any function symbol $f \in \mathscr{F}$ has its *arity* (denoted by $\mathrm{arity}(f)$). We define the set $T(\mathscr{F}, \mathscr{V})$ of *terms* like this: (1) $\mathscr{V} \subseteq T(\mathscr{F}, \mathscr{V})$; and (2) $f(t_1, \ldots, t_n) \subseteq T(\mathscr{F}, \mathscr{V})$ for any $f \in \mathscr{F}$ such that $\mathrm{arity}(f) = n$ and $t_1, \ldots, t_n \in T(\mathscr{F}, \mathscr{V})$. A term without variables is a *ground* term. The set of ground terms is denoted by $T(\mathscr{F})$. A *linear* term is a term in which any variable appears at most once. For any term s, the set of function symbols and variables in s are denoted by $\mathscr{F}(s)$ and $\mathscr{V}(s)$, respectively. For a term $s = f(s_1, \ldots, s_n)$, the *root* symbol of s is f (denoted by $\mathrm{root}(s) = f$). A *substitution* θ is a mapping from $\mathscr{V}$ to $T(\mathscr{F}, \mathscr{V})$. A substitution θ is extended to a mapping $\hat{\theta}$ over term $T(\mathscr{F}, \mathscr{V})$ like this: (1) $\hat{\theta}(x) = \theta(x)$ if $x \in \mathscr{V}$, (2) $\hat{\theta}(f(s_1, \ldots, s_n)) = f(\hat{\theta}(s_1), \ldots, \hat{\theta}(s_n))$. 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 $\mathrm{dom}(\theta) = \{x \in \mathscr{V} \mid x \neq \theta(x)\}$. 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 $T(\mathscr{F} \cup \{\Box_i \mid i \geq 1\}, \mathscr{V})$. $C[s_1, \ldots, s_n]$ 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*). A context C with precisely one hole is denoted by $C[\ ]$. The set of contexts is denoted by $T^{\Box}(\mathscr{F}, \mathscr{V})$; its subset $T(\mathscr{F} \cup \{\Box_i \mid 1 \leq i \leq n\}, \mathscr{V})$ is denoted by $T_n^{\Box}(\mathscr{F}, \mathscr{V})$. $T^{\Box}(\mathscr{F})$ and $T_n^{\Box}(\mathscr{F})$ are defined in the same way as $T(\mathscr{F})$.

A pair $\langle l, r \rangle$ of terms 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 $l \to r$. A *term rewriting system* (*TRS* for short) is a set of rewrite rules. As usual, we always assume that variables in each rewrite rule are disjoint, although the same variable name may be used. A term s *reduces* to a term t by $\mathcal{R}$ (denoted by $s \to_{\mathcal{R}} t$) if there exists 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 $\overset{*}{\leftrightarrow}_{\mathcal{R}}$. A term s is *in normal form* when $s \to_{\mathcal{R}} t$ for no term t. $\mathrm{NF}(\mathcal{R})$ denotes the set of terms in normal form. An *equation* is a pair of terms; we usually write an equation $l \approx r$. For a set $\mathcal{E}$ of equations, we write $s \leftrightarrow_{\mathcal{E}} t$ if there exists 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 reflexive transitive closure of $\leftrightarrow_{\mathcal{E}}$ is denoted by $\overset{*}{\leftrightarrow}_{\mathcal{E}}$.

A rewrite rule $l \to r$ is *left-linear* when l is linear; a TRS $\mathcal{R}$ is left-linear if every rewrite rule in $\mathcal{R}$ is left-linear. 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*. Elements of $T(\mathscr{F}_{\mathrm{c}}, \mathscr{V})$ are called *constructor terms*. A rewrite rule $l \to r$ is a *constructor rule* if $l = f(l_1, \ldots, l_n)$ for some $f \in \mathscr{F}_{\mathrm{d}}$ and $l_1, \ldots, l_n \in T(\mathscr{F}_{\mathrm{c}}, \mathscr{V})$. A TRS $\mathcal{R}$ is a *constructor system* (CS for short) if every rewrite rule is a constructor rule. Without loss of generality, we assume $\mathscr{F}_{\mathrm{d}} = \{\mathrm{root}(l) \mid l \to r \in \mathcal{R}\}$ for a given TRS $\mathcal{R}$. 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 $s_2 \overset{*}{\to}_{\mathcal{R}} t$. Note that $\mathrm{CR}(\mathcal{R})$, $s, t \in \mathrm{NF}(\mathcal{R})$ and $s \overset{*}{\leftrightarrow}_{\mathcal{R}} t$ imply $s = t$. A TRS $\mathcal{R}$ is *strongly normalizing* ($\mathrm{SN}(\mathcal{R})$) if there exists no infinite reduction $s_1 \to_{\mathcal{R}} s_2 \to_{\mathcal{R}} s_3 \to_{\mathcal{R}} \cdots$.

3. Transformation by Templates

This section introduces our framework of program transformation in which programs are formalized by TRSs. Let us start with some motivating examples.

Example 3.1 A program that computes the summation of a list is specified by the following TRS $\mathcal{R}_{sum}$, in which the nat-

ural numbers $0, 1, 2, \ldots$ are expressed as $\mathsf{0}, \mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{s}(\mathsf{0})), \ldots$.

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

This $\mathcal{R}_{sum}$ computes the summation of a list using a recursive call. For instance, $\mathsf{sum}(1{:}(2{:}(3{:}(4{:}(5{:}[\,])))) \xrightarrow{*}_{\mathcal{R}_{sum}} +(1, +(2, +(3, +(4, +(5, \mathsf{sum}([\,])))))) \xrightarrow{*}_{\mathcal{R}_{sum}} 15$.
Using the well-known transformation from the recursive form to the iterative (tail-recursive) form, the following different TRS $\mathcal{R}'_{sum}$ for the list summation program is obtained:

$$\mathcal{R}'_{sum} \left\{ \begin{array}{ll} \mathsf{sum}(x_4) & \to \mathsf{sum1}(x_4, \mathsf{0}) \\ \mathsf{sum1}([\,], x_5) & \to x_5 \\ \mathsf{sum1}(x_6{:}y_6, z_6) & \to \mathsf{sum1}(y_6, +(z_6, x_6)) \\ +(\mathsf{0}, x_7) & \to x_7 \\ +(\mathsf{s}(x_8), y_8) & \to \mathsf{s}(+(x_8, y_8)) \end{array} \right.$$

$\mathcal{R}'_{sum}$ computes the summation of a list more efficiently without the recursion. The equality of the two programs is found using the associativity of the function $+$ and the property $+(\mathsf{0}, n) = +(n, \mathsf{0})$.

Example 3.2 Let us consider another example of program transformation. A program that computes the concatenation of a list of lists is specified by the following TRS $\mathcal{R}_{cat}$.

$$\mathcal{R}_{cat} \left\{ \begin{array}{lcl} \mathsf{cat}([\,]) & \to & [\,] \\ \mathsf{cat}(x_1{:}y_1) & \to & \mathsf{app}(x_1, \mathsf{cat}(y_1)) \\ \mathsf{app}([\,], x_2) & \to & x_2 \\ \mathsf{app}(x_3{:}y_3, z_3) & \to & x_3{:}\mathsf{app}(y_3, z_3) \end{array} \right.$$

For example, we have $\mathsf{cat}([[1,2],[3],[4,5]]) \xrightarrow{*}_{\mathcal{R}_{cat}} [1,2,3,4,5]$. Similarly to Example 3.1, the transformation from the recursive form to the iterative form gives a more efficient TRS $\mathcal{R}'_{cat}$ as follows.

$$\mathcal{R}'_{cat} \left\{ \begin{array}{ll} \mathsf{cat}(x_4) & \to \mathsf{cat1}(x_4, [\,]) \\ \mathsf{cat1}([\,], x_5) & \to x_5 \\ \mathsf{cat1}(x_6{:}y_6, z_6) & \to \mathsf{cat1}(y_6, \mathsf{app}(z_6, x_6)) \\ \mathsf{app}([\,], x_7) & \to x_7 \\ \mathsf{app}(x_8{:}y_8, z_8) & \to x_8{:}\mathsf{app}(y_8, z_8) \end{array} \right.$$

Note that the associativity of the function app and the property $\mathsf{app}([\,], as) = \mathsf{app}(as, [\,])$ hold. Thus, the equality of the two programs is shown similarly.

Example 3.3 One can easily observe that these two transformations in the previous examples can be generalized to a more abstract "transformation template": the *TRS pattern* $\mathcal{P}$

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

is transformed to the TRS pattern $\mathcal{P}'$

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

All the function symbols f, a, b, g, $\cdots$ occurring in the TRS patterns $\mathcal{P}$ and $\mathcal{P}'$ are *pattern variables*. If we match the TRS pattern $\mathcal{P}$ to a concrete TRS $\mathcal{R}$ with an instantiation for these pattern variables, we obtain a more efficient TRS $\mathcal{R}'$ by applying this instantiation to the pattern $\mathcal{P}'$. The equality of $\mathcal{R}_{sum}$ and $\mathcal{R}'_{sum}$ ($\mathcal{R}_{cat}$ and $\mathcal{R}'_{cat}$) is guaranteed when the instantiation satisfies the following equations, called a *hypothesis*:

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

We are now going to introduce a formal definition of a "transformation template".

Definition 3.4 Let $\mathscr{X}$ be a set of pattern variables (disjoint from $\mathscr{F}$ and $\mathscr{V}$) where each pattern variable $p \in \mathscr{X}$ has its arity (denoted by $\mathrm{arity}(p)$). A *pattern* is a term in $T(\mathscr{F} \cup \mathscr{X}, \mathscr{V})$. A *TRS pattern* $\mathcal{P}$ is a set of rewriting rules over patterns. A *hypothesis* $\mathcal{H}$ is a set of equations over patterns. A *transformation template* (or just *template*) is a triple $\langle \mathcal{P}, \mathcal{P}', \mathcal{H} \rangle$ of two TRS patterns $\mathcal{P}$, $\mathcal{P}'$ and a hypothesis $\mathcal{H}$. For patterns s, t, we define $s \to_{\mathcal{P}} t$, $s \leftrightarrow_{\mathcal{H}} t$, etc. similarly for terms.

To achieve program transformation using templates, we need a mechanism to specify how a template is applied to a concrete TRS. For this, we use a variant of the notion of tree homomorphism [6]——we call this a *term homomorphism*.

Definition 3.5 Let φ be a mapping from $\mathscr{X} \cup \mathscr{V}$ to $T^{\square}(\mathscr{X} \cup \mathscr{F}, \mathscr{V})$. We say φ is a *term homomorphism* if the following conditions are satisfied:

(1) $\varphi(p) \in T^{\square}_{\mathrm{arity}(p)}(\mathscr{F})$ for any $p \in \mathrm{dom}_{\mathscr{X}}(\varphi)$,
(2) $\varphi(x) \in \mathscr{V}$ for any $x \in \mathrm{dom}_{\mathscr{V}}(\varphi)$,
(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 $\varphi(x) \neq \varphi(y)$,

where $\mathrm{dom}_{\mathscr{X}}(\varphi) = \{p \in \mathscr{X} \mid \varphi(p) \neq p(\Box_1, \ldots, \Box_{\mathrm{arity}(p)})\}$ and $\mathrm{dom}_{\mathscr{V}}(\varphi) = \{x \in \mathscr{V} \mid \varphi(x) \neq x\}$. A term homomorphism φ is extended to a mapping over $T(\mathscr{F} \cup \mathscr{X}, \mathscr{V})$ as follows:

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

Note that $\varphi(s)$ is a pattern for any pattern s and term homomorphism φ. For a term homomorphism φ and a rewrite rule $l \to r$ (an equation $s \approx t$) over patterns, $\varphi(l \to r)$ ($\varphi(s \approx t)$) is defined by $\varphi(l) \to \varphi(r)$ (resp. $\varphi(s) \approx \varphi(t)$). For a TRS pattern $\mathcal{P}$ and a hypothesis $\mathcal{H}$, $\varphi(\mathcal{P})$ and $\varphi(\mathcal{H})$ are defined by $\varphi(\mathcal{P}) = \{\varphi(l \to r) \mid l \to r \in \mathcal{P}\}$ and $\varphi(\mathcal{H}) = \{\varphi(s \approx t) \mid s \approx t \in \mathcal{H}\}$, respectively.

If $\varphi(\mathcal{P}) = \mathcal{R}$ for some term homomorphism φ, we assume $\mathscr{V}(\mathcal{P}) \cap \mathscr{V}(\mathcal{R}) = \emptyset$ without loss of generality.

We are now going to demonstrate that any term homomorphism preserves reduction. This property of term homomorphisms is proved in a straightforward manner using the injectivity of term homomorphisms. To show this, we extend term homomorphisms φ for substitution θ like this: $\varphi(\theta)(x) = \varphi(\theta(\varphi^{-1}(x)))$, where $\varphi^{-1}(x) = y$ if $y \in \mathrm{dom}_{\mathscr{V}}(\varphi)$, and $\varphi(y) = x$; $\varphi^{-1}(x) = x$ otherwise. Note that since term homomorphism φ is injective on $\mathrm{dom}_{\mathscr{V}}(\varphi)$, one can uniquely define the mapping φ^{-1}.

Lemma 3.6 Let t be a pattern, θ a substitution, and φ a term homomorphism such that $\mathscr{V}(t) \subseteq \mathrm{dom}_{\mathscr{V}}(\varphi)$. Then, $\varphi(t\theta) = \varphi(t)\varphi(\theta)$.

(Proof) The proof proceeds by induction on t.

(1) $t = x \in \mathscr{V}$.
Let $\varphi(x) = y$. Then,
$$\begin{aligned} \varphi(x\theta) &= \varphi(\theta(\varphi^{-1}(y))) \\ &= \varphi(\theta)(y) \\ &= \varphi(x)\varphi(\theta). \end{aligned}$$

(2) $t = f(t_1, \ldots, t_n)$ with $f \in \mathscr{F}$.
Then,
$$\begin{aligned} \varphi(t\theta) &= \varphi(f(t_1\theta, \ldots, t_n\theta)) \\ &= f(\varphi(t_1\theta), \ldots, \varphi(t_n\theta)) \\ &= f(\varphi(t_1)\varphi(\theta), \ldots, \varphi(t_n)\varphi(\theta)) \\ &= f(\varphi(t_1), \ldots, \varphi(t_n))\varphi(\theta) \\ &= \varphi(f(t_1, \ldots, t_n))\varphi(\theta) \\ &= \varphi(t)\varphi(\theta). \end{aligned}$$

(3) $t = p(t_1, \ldots, t_n)$ with $p \in \mathscr{X}$.
Then,
$$\begin{aligned} & \varphi(t\theta) \\ = \ & \varphi(p(t_1, \ldots, t_n)\theta) \\ = \ & \varphi(p(t_1\theta, \ldots, t_n\theta)) \\ = \ & \varphi(p)\langle\varphi(t_1\theta), \ldots, \varphi(t_n\theta)\rangle \\ = \ & \varphi(p)\langle\varphi(t_1)\varphi(\theta), \ldots, \varphi(t_n)\varphi(\theta)\rangle \\ = \ & (\varphi(p)\langle\varphi(t_1), \ldots, \varphi(t_n)\rangle)\varphi(\theta) \\ = \ & (\varphi(p(t_1, \ldots, t_n))\varphi(\theta) \\ = \ & \varphi(t)\varphi(\theta). \end{aligned}$$
(Note that $\mathscr{V}(\varphi(p)) = \emptyset$.) □

Lemma 3.7 Let t be a pattern, $C[\,]$ a context, and φ a term homomorphism. Then, $\varphi(C[t]) = \varphi(C)[\varphi(t), \ldots, \varphi(t)]$.

(Proof) The proof proceeds by induction on the size of $C[\,]$.

(1) $C[\,] = \Box$.
Trivial.

(2) $C[\,] = f(s_1, \ldots, C'[\,], \ldots, s_n)$ with $f \in \mathscr{F}$. Then,
$$\begin{aligned} & \varphi(f(s_1, \ldots, C'[t], \ldots, s_n)) \\ = \ & f(\varphi(s_1), \ldots, \varphi(C'[t]), \ldots, \varphi(s_n))) \\ = \ & f(\varphi(s_1), \ldots, \varphi(C')[\varphi(t), \ldots, \varphi(t)], \\ & \qquad \ldots, \varphi(s_n))) \\ = \ & f(\varphi(s_1), \ldots, \varphi(C'), \ldots, \varphi(s_n)) \\ & \qquad [\varphi(t), \ldots, \varphi(t)] \\ = \ & \varphi(C)[\varphi(t), \ldots, \varphi(t)] \end{aligned}$$

(3) $C[\,] = p(s_1, \ldots, s_n)$ with $s_i = C'[\,]$ and $p \in \mathscr{X}$.
Then,
$$\begin{aligned} & \varphi(p(s_1, \ldots, C'[t], \ldots, s_n)) \\ = \ & \varphi(p)\langle\varphi(s_1), \ldots, \varphi(C'[t]), \ldots, \varphi(s_n)\rangle \\ = \ & \varphi(p)\langle\varphi(s_1), \ldots, \\ & \qquad \varphi(C')[\varphi(t), \ldots, \varphi(t)], \ldots, \varphi(s_n)\rangle \\ = \ & (\varphi(p)\langle\varphi(s_1), \ldots, \varphi(C'), \ldots, \varphi(s_n)\rangle) \\ & \qquad [\varphi(t), \ldots, \varphi(t)] \\ = \ & \varphi(p(s_1, \ldots, C', \ldots, s_n)) \\ & \qquad [\varphi(t), \ldots, \varphi(t)] \\ = \ & \varphi(C)[\varphi(t), \ldots, \varphi(t)]. \end{aligned}$$
□

Proposition 3.8 Let $\mathcal{P}$ be a TRS pattern, $\mathcal{R}$ a TRS, $\mathcal{H}$ a hypothesis, $\mathcal{E}$ a set of equations, and φ a term homomorphism such that $\varphi(\mathcal{P}) = \mathcal{R}$ ($\varphi(\mathcal{H}) = \mathcal{E}$). If $s \to_{\mathcal{P}} t$ ($s \leftrightarrow_{\mathcal{H}} t$), then we have $\varphi(s) \to_{\mathcal{R}} \varphi(t)$ (resp. $\varphi(s) \leftrightarrow_{\mathcal{E}} \varphi(t)$).

(Proof) Suppose $s \to_{\mathcal{P}} t$. Then, there exists a context $C[\,]$, a substitution θ, and a rewrite rule pattern $l \to r \in \mathcal{P}$ such that $s = C[l\theta]$ and $r = C[r\theta]$. Also, $\mathscr{V}(l), \mathscr{V}(r) \subseteq \mathrm{dom}(\varphi)$ by

$\mathcal{V}(\mathcal{P}) \cap \mathcal{V}(\mathcal{R}) = \emptyset$. Then,

$$\begin{aligned}
\varphi(s) &= \varphi(C[l\theta]) \\
&= \varphi(C)[\varphi(l\theta), \ldots, \varphi(l\theta)] \\
&\qquad \text{(by Lemma 3.7)} \\
&= \varphi(C)[\varphi(l)\varphi(\theta), \ldots, \varphi(l)\varphi(\theta)] \\
&\qquad \text{(by Lemma 3.6)} \\
&\xrightarrow{*}_{\mathcal{R}} \varphi(C)[\varphi(r)\varphi(\theta), \ldots, \varphi(r)\varphi(\theta)] \\
&= \varphi(C)[\varphi(r\theta), \ldots, \varphi(r\theta)] \\
&\qquad \text{(by Lemma 3.6)} \\
&= \varphi(C[r\theta]) \text{(by Lemma 3.7)} \\
&= \varphi(t).
\end{aligned}$$

It can be shown that $s \leftrightarrow_{\mathcal{H}} t$ implies $\varphi(s) \leftrightarrow_{\mathcal{E}} \varphi(t)$ in a similar way. □

The TRS transformation by a template is defined as follows.

Definition 3.9 Let $\langle \mathcal{P}, \mathcal{P}', \mathcal{H} \rangle$ be a template. A TRS $\mathcal{R}$ *is transformed into* $\mathcal{R}'$ *by* $\langle \mathcal{P}, \mathcal{P}', \mathcal{H} \rangle$ if there exists a term homomorphism φ such that $\mathcal{R} = \varphi(\mathcal{P}) \cup \mathcal{R}_{com}$ and $\mathcal{R}' = \varphi(\mathcal{P}') \cup \mathcal{R}_{com}$ for some TRS $\mathcal{R}_{com}$.

Note that the hypothesis $\mathcal{H}$ is not used in the definition of the transformation, but it will be needed later when we discuss the *correctness of the transformation*.

Example 3.10 Let $\mathcal{R}_{sum}$, $\mathcal{R}'_{sum}$ be the TRSs in Example 3.1, and $\langle \mathcal{P}, \mathcal{P}', \mathcal{H} \rangle$ the template given in Example 3.3. Then, the following term homomorphism φ satisfies $\mathcal{R}_{sum} = \varphi(\mathcal{P})$ and $\mathcal{R}'_{sum} = \varphi(\mathcal{P}')$.

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

Thus, the TRS $\mathcal{R}_{sum}$ is transformed into $\mathcal{R}'_{sum}$ by $\langle \mathcal{P}, \mathcal{P}', \mathcal{H} \rangle$ where $\mathcal{R}_{com} = \emptyset$.

Example 3.11 Let $\mathcal{R}_{cat}$, $\mathcal{R}'_{cat}$ be the TRSs in Example 3.2, and $\langle \mathcal{P}, \mathcal{P}', \mathcal{H} \rangle$ the template given in Example 3.3. Then, the following term homomorphism φ satisfies $\mathcal{R}_{cat} = \varphi(\mathcal{P})$ and $\mathcal{R}'_{cat} = \varphi(\mathcal{P}')$.

$$\varphi = \left\{ \begin{array}{ll}
\mathsf{f} \mapsto \mathsf{cat}(\Box_1), & u_1 \mapsto x_1, u_6 \mapsto x_6, \\
\mathsf{g} \mapsto \mathsf{app}(\Box_1, \Box_2), & v_1 \mapsto y_1, v_6 \mapsto y_6, \\
\mathsf{f}_1 \mapsto \mathsf{cat1}(\Box_1, \Box_2), & u_2 \mapsto x_2, w_6 \mapsto z_6, \\
\mathsf{a} \mapsto [\,], & v_3 \mapsto y_3, u_7 \mapsto x_7, \\
\mathsf{b} \mapsto [\,], & u_3 \mapsto x_3, u_8 \mapsto x_8, \\
\mathsf{c} \mapsto \Box_1 : \Box_2, & w_3 \mapsto z_3, v_8 \mapsto y_8, \\
\mathsf{d} \mapsto \Box_1 : \Box_2, & u_4 \mapsto x_4, w_8 \mapsto z_8 \\
 & u_5 \mapsto x_5,
\end{array} \right\}$$

Thus, the TRS $\mathcal{R}_{cat}$ is transformed into $\mathcal{R}'_{cat}$ by $\langle \mathcal{P}, \mathcal{P}', \mathcal{H} \rangle$ where $\mathcal{R}_{com} = \emptyset$.

Readers can easily observe from these examples that $\mathcal{R}_{sum}$ and $\mathcal{R}_{cat}$ are respectively transformed into $\mathcal{R}'_{sum}$ and $\mathcal{R}'_{cat}$ in the same way. A question naturally arises from this observation: *does the template guarantee the correctness of all the transformations done by that template?* In the next section, we will discuss the criteria for the templates for the correct transformation and try to give a definite answer to this question.

4. Correctness of the Templates

This section discusses how the correctness of program transformation using templates is validated, i.e., when the equivalence of the input and output programs of program transformations are guaranteed. Intuitively, a program transformation from one program to another is correct if these programs compute the same answer for any input data. In term rewriting, this notion is formalized in the following way.

Definition 4.1 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* $\mathscr{G}$ (notation, $\mathcal{R} \simeq_{\mathscr{G}} \mathcal{R}'$), if for any ground term $s \in T(\mathscr{G})$ and ground constructor term $t \in T(\mathscr{F}_{\mathrm{c}})$, $s \xrightarrow{*}_{\mathcal{R}} t$ iff $s \xrightarrow{*}_{\mathcal{R}'} t$ holds.

At this juncture, we need to make a short remark about the definition of the equivalence of TRSs. In a program transformation from $\mathcal{R}$ to $\mathcal{R}'$, one cannot generally expect $s \xrightarrow{*}_{\mathcal{R}} t$ iff $s \xrightarrow{*}_{\mathcal{R}'} t$ for all ground terms $s \in T(\mathscr{F})$ and ground constructor term $t \in T(\mathscr{F}_{\mathrm{c}})$. This is because one TRS may use some subfunctions that the other may not have. This is why the equivalence of TRSs is defined with respect to a set $\mathscr{G}$ of function symbols. Intuitively, the functions in $\mathscr{G}$ are those originally required to compute by the TRSs in comparison.

Example 4.2 Let us consider $\mathcal{R}_{sum}$ and $\mathcal{R}'_{sum}$ in Example 3.1. Then, $\mathsf{sum1}([\,], \mathsf{s}(0)) \to_{\mathcal{R}'_{sum}} \mathsf{s}(0) \in T(\mathscr{F}_{\mathrm{c}})$, but $\mathsf{sum1}([\,], \mathsf{s}(0))$ is in a normal form of $\mathcal{R}_{sum}$, because $\mathcal{R}_{sum}$ has no rewrite rules for $\mathsf{sum1}$. Thus, $\mathcal{R}_{sum} \not\simeq_{\mathscr{G}} \mathcal{R}'_{sum}$ for any $\mathscr{G}$ containing $\mathsf{sum1}$. Rather, we should consider the equivalence of these TRSs by setting $\mathscr{G} = \{\mathsf{sum}, +, :, [\,], \mathsf{s}, 0\}$; indeed, in that case we can prove $\mathcal{R}_{sum} \simeq_{\mathscr{G}} \mathcal{R}'_{sum}$.

Although whether two TRSs are equivalent cannot generally be decided, it is known that two TRSs are equivalent when there exists

an *equivalent transformation* from one to the other[18] for some restricted class of TRSs. Let us now simplify and improve this technique for our framework.

For a set $\mathscr{G}$ of function symbols, we speak of a TRS $\mathcal{R}$ (or a set $\mathcal{E}$ of equations) *over* $\mathscr{G}$ when all rewrite rules (resp. equations) consist of terms in $T(\mathscr{G}, \mathscr{V})$.

Definition 4.3 Let $\mathcal{R}_0$ be a left-linear CS over $\mathscr{F}_0$ and $\mathcal{E}$ be 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 $f(x_1, \ldots, x_n) \to r$ is a left-linear constructor rewrite rule such that $f \notin \mathscr{F}_k$ and $r \in T(\mathscr{F}_k, \mathscr{V})$. 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*
$$\mathcal{R}_{k+1} = \mathcal{R}_k \setminus \{l \to r\}$$

When 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* $\mathcal{E}$ when there exits an equivalent transformation sequence $\mathcal{R} \overset{*}{\underset{I}{\Rightarrow}} \cdot \overset{*}{\underset{A}{\Rightarrow}} \cdot \overset{*}{\underset{E}{\Rightarrow}} \mathcal{R}'$ under $\mathcal{E}$.

To state the criteria for the equivalence of TRSs based on equivalence transformation, we introduce some standard notions related to proving the inductive theorem proving in what follows.

Suppose $\mathscr{F}_c \subseteq \mathscr{G} \subseteq \mathscr{F}$. A TRS $\mathcal{R}$ is *sufficiently complete for* $\mathscr{G}$ ($\mathrm{SC}(\mathcal{R}, \mathscr{G})$) when for any ground term $s \in T(\mathscr{G})$ there exists $t \in T(\mathscr{F}_c)$ such that $s \overset{*}{\to}_{\mathcal{R}} t$. A substitution θ is *ground on* $\mathscr{G}$ if $\theta(x) \in T(\mathscr{G})$ for any $x \in \mathrm{dom}(\theta)$. An equation $s \approx t$ is an *inductive consequence of* $\mathcal{R}$ *for* $\mathscr{G}$ ($\mathcal{R}, \mathscr{G} \vdash_{ind} s \approx t$) when 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 $s \approx t \in \mathcal{E}$.

Theorem 4.4 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 $\mathcal{R} \simeq_{\mathscr{G} \cap \mathscr{G}'} \mathcal{R}'$.

(Proof) Suppose $\mathcal{R} \overset{*}{\underset{I}{\Rightarrow}} \mathcal{R}_I \overset{*}{\underset{A}{\Rightarrow}} \mathcal{R}_A \overset{*}{\underset{E}{\Rightarrow}} \mathcal{R}'$. We first show some properties of $\mathcal{R}_I$. Let $\mathcal{R}_0 = \mathcal{R}$ and $\mathcal{R}_i \underset{I}{\Rightarrow} \mathcal{R}_i \cup \{f(x_1, \ldots, x_n) \to r\} = \mathcal{R}_{i+1}$. Then, $\mathrm{SC}(\mathcal{R}_i, \mathscr{F}_i)$ implies $\mathrm{SC}(\mathcal{R}_{i+1}, \mathscr{F}_i \cup \{f\})$ by the definition of the Introduction rule. Thus, by our assumption $\mathrm{SC}(\mathcal{R}, \mathscr{G})$, it easily follows by induction on the length of $\mathcal{R} \overset{*}{\underset{I}{\Rightarrow}} \mathcal{R}_i$ that $\mathrm{SC}(\mathcal{R}_i, \mathscr{F}_i)$ for all i such that $\mathcal{R} \overset{*}{\underset{I}{\Rightarrow}} \mathcal{R}_i$. Thus, we may assume w.l.o.g. $\mathrm{SC}(\mathcal{R}_I, \mathscr{F})$, because we may ignore any function symbols not appearing even in $\mathcal{R}_I$. It is clear that $\mathcal{R} \subseteq \mathcal{R}_I$ by the definition of the Introduction rule. Also, from $\mathrm{CR}(\mathcal{R}_0)$ and the fact that each introduced rewrite rule $f(x_1, \ldots, x_n) \to r$ at $i+1$ is left-linear and non-overlapping with left-linear TRS $\mathcal{R}_i$, it follows that $\mathrm{CR}(\mathcal{R}_I)$ using the commutativity of TRSs[17),18]. Thus, for $\mathcal{R}_I$, we have (1) $\mathrm{SC}(\mathcal{R}_I, \mathscr{F})$, (2) $\mathcal{R} \subseteq \mathcal{R}_I$, and (3) $\mathrm{CR}(\mathcal{R}_I)$. We next show that $\overset{*}{\leftrightarrow}_{\mathcal{R}} = \overset{*}{\leftrightarrow}_{\mathcal{R}'}$ on $T(\mathscr{G} \cap \mathscr{G}')$.

(1) $\overset{*}{\leftrightarrow}_{\mathcal{R}} = \overset{*}{\leftrightarrow}_{\mathcal{R}_I}$ on $T(\mathscr{G})$. (i.e., for any $s, t \in T(\mathscr{G})$, $s \overset{*}{\leftrightarrow}_{\mathcal{R}} t$ iff $s \overset{*}{\leftrightarrow}_{\mathcal{R}_I} t$.)
($\subseteq$) Trivial. ($\supseteq$) Suppose that $s \overset{*}{\leftrightarrow}_{\mathcal{R}_I} t$ where $s, t \in T(\mathscr{G})$. By $\mathrm{SC}(\mathcal{R}, \mathscr{G})$, there exist ground constructor terms $s', t' \in T(\mathscr{F}_c)$ such that $s \overset{*}{\to}_{\mathcal{R}} s'$ and $t \overset{*}{\to}_{\mathcal{R}} t'$. From $\mathcal{R} \subseteq \mathcal{R}_I$, we have $s \overset{*}{\to}_{\mathcal{R}_I} s'$ and $t \overset{*}{\to}_{\mathcal{R}_I} t'$. Thus, by $\mathrm{CR}(\mathcal{R}_I)$ and $T(\mathscr{F}_c) \subseteq \mathrm{NF}(\mathcal{R}_I)$, $s' = t'$ holds. This means $s \overset{*}{\to}_{\mathcal{R}} s' = t' \overset{*}{\leftarrow}_{\mathcal{R}} t$.

(2) $\overset{*}{\leftrightarrow}_{\mathcal{R}_I} = \overset{*}{\leftrightarrow}_{\mathcal{R}_A}$ on $T(\mathscr{F})$. (i.e., for any $s, t \in T(\mathscr{F})$, $s \overset{*}{\leftrightarrow}_{\mathcal{R}_I} t$ iff $s \overset{*}{\leftrightarrow}_{\mathcal{R}_A} t$.)
($\subseteq$) Trivial. ($\supseteq$) Suppose that $s \leftrightarrow_{\mathcal{E}} t$ where $s, t \in T(\mathscr{F})$. By the definition of $\leftrightarrow_{\mathcal{E}}$, there exist a context $C[\ ]$, a ground substitution θ_g, and an equation $l \approx r \in \mathcal{E}$ or $r \approx l \in \mathcal{E}$ such that $s = C[l\theta_g]$ and $t = C[r\theta_g]$. By $\mathrm{SC}(\mathcal{R}_I, \mathscr{F})$, there exists a ground substitution θ_g^c such that $\theta_g(x) \overset{*}{\to}_{\mathcal{R}_I} \theta_g^c(x) \in T(\mathscr{F}_c)$ for any $x \in \mathrm{dom}(\theta_g)$. Then, $C[l\theta_g] \overset{*}{\to}_{\mathcal{R}_I} C[l\theta_g^c]$ and $C[r\theta_g] \overset{*}{\to}_{\mathcal{R}_I} C[r\theta_g^c]$ hold. Now, since $l\theta_g^c, r\theta_g^c \in T(\mathscr{G})$, we have $l\theta_g^c \overset{*}{\leftrightarrow}_{\mathcal{R}} r\theta_g^c$

by our assumption $\mathcal{R}, \mathscr{G} \vdash_{ind} \mathcal{E}$. Thus, by $\mathcal{R} \subseteq \mathcal{R}_I$, $C[l\theta_g^c] \overset{*}{\leftrightarrow}_{\mathcal{R}_I} C[r\theta_g^c]$ holds. Hence, $\overset{*}{\leftrightarrow}_{\mathcal{R}_I} \supseteq \leftrightarrow_{\mathcal{E}}$ on $\mathscr{T}$. It is easy to see by the definition of the Addition rule that $\overset{*}{\leftrightarrow}_{\mathcal{R}_A} = \overset{*}{\leftrightarrow}_{\mathcal{E} \cup \mathcal{R}_I}$ on $T(\mathscr{F}, \mathscr{V})$. Hence, $\overset{*}{\leftrightarrow}_{\mathcal{R}_A} \subseteq \overset{*}{\leftrightarrow}_{\mathcal{E} \cup \mathcal{R}_I} \subseteq \overset{*}{\leftrightarrow}_{\mathcal{R}_I}$ on $T(\mathscr{F})$.

(3) $\overset{*}{\leftrightarrow}_{\mathcal{R}_I} = \overset{*}{\leftrightarrow}_{\mathcal{R}'}$ on $T(\mathscr{G}')$ (i.e., for any $s, t \in T(\mathscr{G}')$, $s \overset{*}{\leftrightarrow}_{\mathcal{R}_I} t$ iff $s \overset{*}{\leftrightarrow}_{\mathcal{R}'} t$.)
($\supseteq$) It easily follows from item 2 and the definition of the Elimination rule. ($\subseteq$) Suppose that $s \overset{*}{\leftrightarrow}_{\mathcal{R}_I} t$ where $s, t \in T(\mathscr{G}')$. From $\mathrm{SC}(\mathcal{R}', \mathscr{G}')$, there exist ground constructor terms $s', t' \in T(\mathscr{F}_c)$ such that $s \overset{*}{\to}_{\mathcal{R}'} s'$ and $t \overset{*}{\to}_{\mathcal{R}'} t'$. As we have already shown $\overset{*}{\leftrightarrow}_{\mathcal{R}_I} \supseteq \overset{*}{\leftrightarrow}_{\mathcal{R}'}$ on $T(\mathscr{G}')$, it follows that $s' \overset{*}{\leftrightarrow}_{\mathcal{R}_I} s \overset{*}{\leftrightarrow}_{\mathcal{R}_I} t \overset{*}{\leftrightarrow}_{\mathcal{R}_I} t'$. From $\mathrm{CR}(\mathcal{R}_I)$ and $T(\mathscr{F}_c) \subseteq \mathrm{NF}(\mathcal{R}_I)$, $s' = t'$ holds. This means $s \overset{*}{\to}_{\mathcal{R}'} s' = t' \overset{*}{\leftarrow}_{\mathcal{R}'} t$.

From 1, 3, and $T(\mathscr{G} \cap \mathscr{G}') \subseteq T(\mathscr{G}), T(\mathscr{F}), T(\mathscr{G}')$, it follows that $\overset{*}{\leftrightarrow}_{\mathcal{R}} = \overset{*}{\leftrightarrow}_{\mathcal{R}'}$ on $T(\mathscr{G} \cap \mathscr{G}')$. Finally, we show $\mathcal{R} \simeq_{\mathscr{G} \cap \mathscr{G}'} \mathcal{R}'$. Suppose $s \in T(\mathscr{G} \cap \mathscr{G}')$, $t \in T(\mathscr{F}_c)$, and $s \overset{*}{\to}_{\mathcal{R}} t$. From $\mathrm{SC}(\mathcal{R}', \mathscr{G}')$, there exists a constructor term $t' \in T(\mathscr{F}_c)$ such that $s \overset{*}{\to}_{\mathcal{R}'} t'$. By $\overset{*}{\leftrightarrow}_{\mathcal{R}} = \overset{*}{\leftrightarrow}_{\mathcal{R}'}$ on $T(\mathscr{G} \cap \mathscr{G}')$ and $\mathscr{F}_c \subseteq \mathscr{G} \cap \mathscr{G}'$, we have $t \overset{*}{\leftrightarrow}_{\mathcal{R}} t'$. Then, by $\mathrm{CR}(\mathcal{R})$ and $T(\mathscr{F}_c) \subseteq \mathrm{NF}(\mathcal{R})$, it follows $t = t'$. Hence, $s \overset{*}{\to}_{\mathcal{R}'} t' = t$. Conversely, suppose $s \in T(\mathscr{G} \cap \mathscr{G}')$, $t \in T(\mathscr{F}_c)$, and $s \overset{*}{\to}_{\mathcal{R}'} t$. From $\mathrm{SC}(\mathcal{R}, \mathscr{G})$, there exists a constructor term $t' \in T(\mathscr{F}_c)$ such that $s \overset{*}{\to}_{\mathcal{R}} t'$. By $\overset{*}{\leftrightarrow}_{\mathcal{R}} = \overset{*}{\leftrightarrow}_{\mathcal{R}'}$ on $T(\mathscr{G} \cap \mathscr{G}')$ and $\mathscr{F}_c \subseteq \mathscr{G} \cap \mathscr{G}'$, we have $t \overset{*}{\leftrightarrow}_{\mathcal{R}} t'$. Then, by $\mathrm{CR}(\mathcal{R})$ and $T(\mathscr{F}_c) \subseteq \mathrm{NF}(\mathcal{R})$, it follows $t = t'$. Hence, $s \overset{*}{\to}_{\mathcal{R}} t' = t$. □

Example 4.5 Let $\mathcal{R}_{sum}, \mathcal{R}'_{sum}$ be the TRSs in Example 3.1. Let $\mathcal{E}$ be the following set of equations.

$$\mathcal{E} \quad \begin{cases} +(\mathsf{0}, x) & \approx & +(x, \mathsf{0}) \\ +(+(x, y), z) & \approx & +(x, +(y, z)) \end{cases}$$

Note that any equation in $\mathcal{E}$ is an inductive consequence of $\mathcal{R}_{sum}$ for $\mathscr{G} = \{\mathsf{sum}, +, :, [\,], \mathsf{s}, \mathsf{0}\}$, i.e., $\mathcal{R}, \mathscr{G} \vdash_{ind} \mathcal{E}$.

We now demonstrate an equivalent transformation from $\mathcal{R}_{sum}$ to $\mathcal{R}'_{sum}$ under $\mathcal{E}$. Let $\mathcal{R}_0 = \mathcal{R}_{sum}$.

(1) Let $\mathcal{R}_1 = \mathcal{R}_0 \cup \{\mathsf{sum}_1(x, y) \to +(y, \mathsf{sum}(x))\}$. Clearly, $\mathcal{R}_0 \Rightarrow \mathcal{R}_1$ by the Introduction rule.

(2) Let $\mathcal{R}_2 = \mathcal{R}_1 \cup \{\mathsf{sum}(x) \to \mathsf{sum}_1(x, \mathsf{0})\}$. Here, we have

$$\begin{array}{lll} \mathsf{sum}(x) & \leftarrow_{\mathcal{R}_1} & +(\mathsf{0}, \mathsf{sum}(x)) \\ & \leftarrow_{\mathcal{R}_1} & \mathsf{sum}_1(x, \mathsf{0}) \end{array}$$

Thus, $\mathcal{R}_1 \Rightarrow \mathcal{R}_2$ by the Addition rule.

(3) Let $\mathcal{R}_3 = \mathcal{R}_2 \cup \{\mathsf{sum}_1([\,], x) \to x\}$. Then, we have

$$\begin{array}{lll} \mathsf{sum}_1([\,], x) & \to_{\mathcal{R}_2} & +(x, \mathsf{sum}([\,])) \\ & \to_{\mathcal{R}_2} & +(x, \mathsf{0}) \\ & \leftrightarrow_{\mathcal{E}} & +(\mathsf{0}, x) \\ & \to_{\mathcal{R}_2} & x \end{array}$$

Thus, $\mathcal{R}_2 \Rightarrow \mathcal{R}_3$ by the Addition rule.

(4) Let $\mathcal{R}_4 = \mathcal{R}_3 \cup \{\mathsf{sum}_1(x{:}y, z) \to \mathsf{sum}_1(y, +(z, x))\}$. Then, we have

$$\begin{array}{lll} \mathsf{sum}_1(x{:}y, z) & \to_{\mathcal{R}_3} & +(z, \mathsf{sum}(x{:}y)) \\ & \to_{\mathcal{R}_3} & +(z, +(x, \mathsf{sum}(y))) \\ & \leftrightarrow_{\mathcal{E}} & +(+(z, x), \mathsf{sum}(y)) \\ & \leftarrow_{\mathcal{R}_3} & \mathsf{sum}_1(y, +(z, x)) \end{array}$$

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

(5) Finally, applying the Elimination rule three times to $\mathcal{R}_4$, we obtain $\mathcal{R}'_{sum}$.

Thus, there exists an equivalent transformation from $\mathcal{R}_{sum}$ to $\mathcal{R}'_{sum}$ under $\mathcal{E}$. It is easily shown that $\mathcal{R}_{sum}$ is confluent and sufficiently complete for $\mathscr{G}$ and that $\mathcal{R}'_{sum}$ is sufficiently complete for $\mathscr{G} \cup \{\mathsf{sum1}\}$. Therefore, from Theorem 4.4, it follows that $\mathcal{R}_{sum} \simeq_{\mathscr{G}} \mathcal{R}'_{sum}$.

For the TRS transformation in Example 3.2, it is easily observed that the correctness of the transformation can be proved exactly in the same way. Thus, one may naturally expect that such manual transformations can be conducted at the template level. A naive method of proving the correctness of a template $\langle \mathcal{P}, \mathcal{P}', \mathcal{H} \rangle$ is to find an equivalent transformation from $\mathcal{P}$ to $\mathcal{P}'$ under $\mathcal{H}$ similar to TRSs. This naive method, however, does not work because $\mathcal{P} \overset{*}{\Rightarrow} \mathcal{P}'$ under $\mathcal{H}$ does not imply $\varphi(\mathcal{P}) \overset{*}{\Rightarrow} \varphi(\mathcal{P}')$ under $\varphi(\mathcal{H})$ in general. For example, suppose $\mathcal{P} = \{\mathsf{p}(x) \to \mathsf{a}(\mathsf{b})\}$ and $\mathcal{P}' = \mathcal{P} \cup \{\mathsf{q}(x) \to \mathsf{b}\}$. Then, $\mathcal{P} \underset{I}{\Rightarrow} \mathcal{P}'$. However, $\varphi(\mathcal{P}) \not\underset{I}{\Rightarrow} \varphi(\mathcal{P}')$ when $\varphi = \{\mathsf{p} \mapsto \mathsf{f}(\square_1), \mathsf{q} \mapsto \mathsf{f}(\square_1), \mathsf{a} \mapsto \mathsf{0}, \mathsf{b} \mapsto \mathsf{1}\}$. The key idea in the proof of Theorem 4.4 is the preservation of the Church-Rosser property under the Introduction rule. In the example above, $\varphi(\mathcal{P}')$ does not have the Church-Rosser property even though $\mathcal{P}'$ does. Thus, in order to preserve the correctness of each step, in particular the *Introduction* step in equivalence transformation, some restrictions on the term homomorphism φ are necessary. This leads us to the notion of CS homomorphisms.

Similar to the set $\mathscr{F}$ of function symbols, we assume that the set $\mathscr{X}$ of pattern variables is divided into two disjoint sets——the set $\mathscr{X}_{\rm d}$ of *defined pattern variables* and $\mathscr{X}_{\rm c}$ of *constructor pattern variables*. All the notions concerning constructor TRSs are then naturally extended to TRS patterns.

Definition 4.6 A term homomorphism φ is a *CS homomorphism* if for any defined pattern variable p, $\varphi(p) = f(\Box_{i_1}, \ldots, \Box_{i_n})$ for some defined function symbol f and mutually distinct indexes $i_1, \ldots, i_n$, and (2) $\mathrm{root}(\varphi(p)) = \mathrm{root}(\varphi(q))$ implies $p = q$ for any defined pattern variables p and q.

We now propose inference rules to develop a template for correct transformations. Once a template is obtained with these inference rules then the correctness of all the TRS transformations obtained by this template are verified *automatically*.

Definition 4.7 Let $\mathcal{P}_0$ be a left-linear CS pattern over $\mathscr{X}_0$ and $\mathcal{H}$ a set of equations over $\mathscr{X}_0$. An equivalent transformation sequence under $\mathcal{H}$ is a sequence $\mathcal{P}_0, \ldots, \mathcal{P}_n$ of CS patterns (over $\mathscr{X}_0, \ldots, \mathscr{X}_n$, 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 $p(x_1, \ldots, x_n) \to r$ is a left-linear rewrite rule such that $p \in \mathscr{X}_{\rm d} \setminus \mathscr{X}_k$ and $r \in T(\mathscr{X}_k \cap \mathscr{X}_{\rm d}, \mathscr{V})$. We put $\mathscr{X}_{k+1} = \mathscr{X}_k \cup \{p\}$.

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

(In Addition and Elimination rules, $\mathscr{X}_{k+1}$ can be any set of pattern variables such that $\mathscr{X}_{k+1} \subseteq \mathscr{X}_k$ provided that $\mathcal{P}_{k+1}$ is a TRS pattern over $\mathscr{X}_{k+1}$.) Similar to the equivalence transformation of TRSs, we say there exists an *equivalent transformation from $\mathcal{P}$ to $\mathcal{P}'$ under $\mathcal{H}$* when there exits an equivalent transformation sequence $\mathcal{P} \overset{*}{\underset{I}{\Rightarrow}} \cdot \overset{*}{\underset{A}{\Rightarrow}} \cdot \overset{*}{\underset{E}{\Rightarrow}} \mathcal{P}'$ under $\mathcal{H}$.

Definition 4.8 A template $\langle \mathcal{P}, \mathcal{P}', \mathcal{H} \rangle$ is *developed* if there exists an equivalent transformation from $\mathcal{P}$ to $\mathcal{P}'$ under $\mathcal{H}$.

Definition 4.9 Let $\mathscr{G}$ be a set of function symbols such that $\mathscr{F}_{\rm c} \subseteq \mathscr{G} \subseteq \mathscr{F}$ and $\mathcal{R}$ a left-linear CS over $\mathscr{G}$. Then, $\mathcal{R}$ *is transformed into a TRS $\mathcal{R}'$ w.r.t. $\mathscr{G}$ by a template* $\langle \mathcal{P}, \mathcal{P}', \mathcal{H} \rangle$ if there exists a CS homomorphism φ such that $\mathcal{R} = \varphi(\mathcal{P}) \cup \mathcal{R}_{com}$, $\mathcal{R}' = \varphi(\mathcal{P}') \cup \mathcal{R}_{com}$, $\mathcal{R}, \mathscr{G} \vdash_{ind} \varphi(\mathcal{H})$, and $\mathrm{root}(\varphi(p)) \notin \mathscr{F}_{\rm d}(\mathcal{R}_{com})$ for any $p \in \mathscr{X}_{\rm d} \setminus \mathscr{X}_{\rm d}(\mathcal{P})$.

To give criteria for the correctness of the TRS transformation with the developed templates, we now prepare a couple of lemmas.

Lemma 4.10 Let φ be a CS homomorphism such that $\mathrm{root}(\varphi(p)) \notin \mathscr{F}_{\rm d}(\mathcal{R}_{com})$ for any $p \in \mathscr{X}_{\rm d} \setminus \mathscr{X}_{\rm d}(\mathcal{P})$. If $\mathcal{P} \underset{I}{\Rightarrow} \mathcal{P}'$ then $\varphi(\mathcal{P}) \cup \mathcal{R}_{com} \underset{I}{\Rightarrow} \varphi(\mathcal{P}') \cup \mathcal{R}_{com}$.

(Proof) Suppose $\mathcal{P}' = \mathcal{P} \cup \{p(x_1, \ldots, x_n) \to r\}$. It is easy to see by the definition of CS homomorphism and the Introduction rule, and the assumption that $\mathrm{root}(\varphi(p))$ does not occur in $\varphi(\mathcal{P}) \cup \mathcal{R}_{com}$, that $\varphi(p(x_1, \ldots, x_n))$ is linear, and that all the function symbols in $\varphi(r)$ occur in $\varphi(\mathcal{P}) \cup \mathcal{R}_{com}$ due to $r \in T(\mathscr{X} \cap \mathscr{X}_{\rm d})$. □

Lemma 4.11 Let φ be a CS homomorphism. If $\mathcal{P} \underset{A}{\Rightarrow} \mathcal{P}'$, then $\varphi(\mathcal{P}) \cup \mathcal{R}_{com} \underset{A}{\Rightarrow} \varphi(\mathcal{P}') \cup \mathcal{R}_{com}$.

(Proof) This immediately follows from Proposition 3.8. □

Thus, we arrive at:

Theorem 4.12 Let $\mathscr{G}$ and $\mathscr{G}'$ be sets of function symbols such that $\mathscr{F}_{\rm c} \subseteq \mathscr{G}, \mathscr{G}' \subseteq \mathscr{F}$. 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 $\mathcal{R}$ is transformed into a TRS $\mathcal{R}'$ w.r.t. $\mathscr{G}$ by a developed template $\langle \mathcal{P}, \mathcal{P}', \mathcal{H} \rangle$. Then, $\mathrm{CR}(\mathcal{R}) \wedge \mathrm{SC}(\mathcal{R}, \mathscr{G}) \wedge \mathrm{SC}(\mathcal{R}', \mathscr{G}')$ imply $\mathcal{R} \simeq_{\mathscr{G} \cap \mathscr{G}'} \mathcal{R}'$.

(Proof) By Lemmas 4.10 and 4.11, and Theorem 4.4. □

Example 4.13 We now demonstrate how a template is developed. Let $\langle \mathcal{P}, \mathcal{P}', \mathcal{H} \rangle$ be the transformation template in Example 3.3. We show there exists an equivalent transformation from $\mathcal{P}$ to $\mathcal{P}'$ under $\mathcal{H}$.

(1) Let $\mathcal{P}_0 = \mathcal{P}$.

(2) Let $\mathcal{P}_1 = \mathcal{P}_0 \cup \{\mathsf{f}_1(u, v) \to \mathsf{g}(v, \mathsf{f}(u))\}$. Here, f_1 is a fresh pattern variable. Thus, $\mathcal{P}_0 \Rightarrow \mathcal{P}_1$ by the Introduction rule.

(3) Let $\mathcal{P}_2 = \mathcal{P}_1 \cup \{\mathsf{f}(u) \to \mathsf{f}_1(u, \mathsf{b})\}$. Here, we have
$$\begin{aligned} \mathsf{f}(u) &\leftarrow_{\mathcal{P}_1} \mathsf{g}(\mathsf{b}, \mathsf{f}(u)) \\ &\leftarrow_{\mathcal{P}_1} \mathsf{f}_1(u, \mathsf{b}). \end{aligned}$$
Thus, $\mathcal{P}_1 \Rightarrow \mathcal{P}_2$ by the Addition rule.

(4) Let $\mathcal{P}_3 = \mathcal{P}_2 \cup \{\mathsf{f}_1(\mathsf{a}, u) \to u\}$. Here, we have

```
FUNCTIONS
 sum:  List -> Nat;
 cons:  Nat * List -> List;
 nil:  List;
 +:  Nat * Nat -> Nat;
 s:  Nat -> Nat;
 0:  Nat

RULES
 sum(nil()) -> 0();
 sum(cons(x,ys)) -> +(x,sum(ys));
 +(0(), x) -> x;
 +(s(x),y) -> s(+(x,y))

INPUT
 ?f(?a()) -> ?b();
 ?f(?c(u,v)) -> ?g(u,?f(v));
 ?g(?b(),u) -> u;
 ?g(?d(u,v),w) -> ?d(u,?g(v,w))

OUTPUT
 ?f(u) -> ?f1(u,?b());
 ?f1(?a(),u) -> u;
 ?f1(?c(u,v),w) -> ?f1(v,?g(w,u));
 ?g(?b(),u) -> u;
 ?g(?d(u,v),w) -> ?d(u,?g(v,w))

HYPOTHESIS
 ?g(?b(),u) = ?g(u,?b());
 ?g(?g(u,v),w) = ?g(u,?g(v,w))
```

Fig. 1 Specification for input TRS and transformation template.

$$\begin{aligned} \mathsf{f}_1(\mathsf{a}, u) &\to_{\mathcal{P}_2} \mathsf{g}(u, \mathsf{f}(\mathsf{a})) \\ &\to_{\mathcal{P}_2} \mathsf{g}(u, \mathsf{b}) \\ &\leftrightarrow_{\mathcal{H}} \mathsf{g}(\mathsf{b}, u) \\ &\to_{\mathcal{P}_2} u \end{aligned}$$

Thus, we have $\mathcal{P}_2 \Rightarrow \mathcal{P}_3$ by the Addition rule.

(5) Let $\mathcal{P}_4 = \mathcal{P}_3 \cup \{\mathsf{f}_1(\mathsf{c}(u, v), w) \to \mathsf{f}_1(v, \mathsf{g}(w, u))\}$. Here, we have

$$\begin{aligned} \mathsf{f}_1(\mathsf{c}(u, v), w) &\to_{\mathcal{P}_3} \mathsf{g}(w, \mathsf{f}(\mathsf{c}(u, v))) \\ &\to_{\mathcal{P}_3} \mathsf{g}(w, \mathsf{g}(u, \mathsf{f}(v))) \\ &\leftrightarrow_{\mathcal{H}} \mathsf{g}(\mathsf{g}(w, u), \mathsf{f}(v)) \\ &\leftarrow_{\mathcal{P}_3} \mathsf{f}_1(v, \mathsf{g}(w, u)) \end{aligned}$$

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

(6) Finally, applying the Elimination rules three times to $\mathcal{P}_4$, we obtain $\mathcal{P}'$.

Thus, the template $\langle \mathcal{P}, \mathcal{P}', \mathcal{H} \rangle$ is developed.

5. Program Transformation System RAPT

This section introduces the program transformation system RAPT (Rewriting-based Automated Program Transformation system) that implements our framework [3].

The source code for RAPT consists of about 5,000 lines and is written in the Standard ML of New Jersey, which is an implementation of the strongly typed functional programming language ML.

RAPT transforms a given many-sorted TRS according to a given transformation template and output it when the equality between the input TRS and obtained TRS is verified. Note that given transformation templates have to be developed to guarantee the correctness of transformation. The extension to many-sorted frameworks is crucial to verifying sufficient completeness; all of our results in the previous sections can easily be adapted to many-sorted frameworks.

Figure 1 shows an example of the input of RAPT. The `FUNCTIONS` and `RULES` sections describe input many-sorted TRS and `INPUT`, `OUTPUT` and `HYPOTHESIS` sections give the transformation template. Figure 1 describes the type information for the $\mathcal{R}_{sum}$, the TRS $\mathcal{R}_{sum}$, and the template that appears in Example 3.3.

The TRS transformation and the verification of the correctness of the transformation is conducted in RAPT by the following six phases:

(1) **Validation of Input TRS**
Checking whether the input TRS is a left-linear constructor system and well-typed.

(2) **Precedence Detection**
Verifying whether the input TRS terminates using lexicographic path ordering via precedence detection [10].

(3) **Proving Confluence and Sufficient Completeness**
Verifying whether the input TRS is confluent and sufficiently complete. Since the termination of the input TRS has already been verified, RAPT only checks the joinability of critical pairs and quasi-reducibility of the TRS

(4) **TRS Pattern Matching**
Finding CS homomorphism from the TRS pattern to TRS, using the pattern

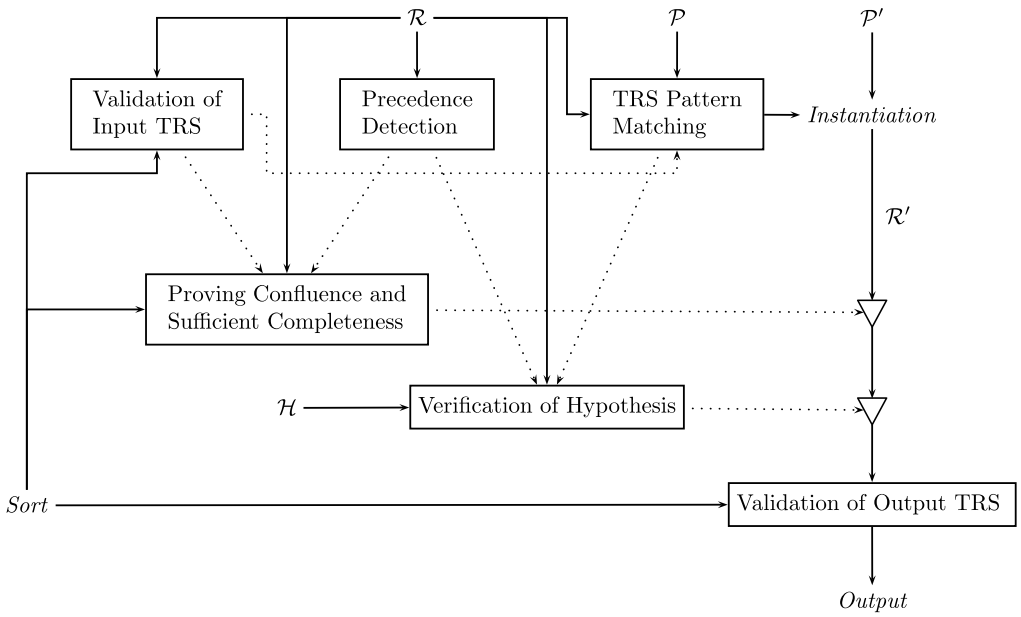

Fig. 2 Overview of RAPT.

matching algorithm **Match** [4].

(5) **Verification of Hypothesis**
Applying the CS homomorphism to the hypothesis of the template and proving whether it is inductive consequences of the input TRS using rewriting induction [14].

(6) **Validation of the Output TRS**
Checking whether the output TRS terminates, is left-linear, well-typed, and sufficiently complete.

Figure 2 describes the dependencies between the six phases. If these six phases are successfully passed then RAPT produces output TRSs. The correctness of the transformation is guaranteed in the sense of Definition 4.1, provided that the template is developed.

The following is the output of RAPT for the run of Fig. 1.

```
[ sum(u) -> f1(u, 0()),
f1(nil(), u) -> u,
f1(cons(u, v), w) -> f1(v, +(w, u)),
+(0(), u) -> u,
+(s(v), w) -> s(+(v, w)) ]
```

The above TRS corresponds to the output TRS $\mathcal{R}'_{sum}$.

6. Examples

This section introduces several developed templates and describes examples of transformations obtaind by these templates. Each transformation can be done in less than 100 msec with RAPT.

6.1 A Transformation from Recursive form to Iterative Form

We considered transformations from recursive programs to iterative in Section 3. The template $\langle \mathcal{P}, \mathcal{P}', \mathcal{H} \rangle$ in Example 3.3 is obtained by generalizing such transformations. We have seen that the transformation of the TRS $\mathcal{R}_{sum}$ and $\mathcal{R}_{cat}$. Let us try to transform other TRSs using $\langle \mathcal{P}, \mathcal{P}', \mathcal{H} \rangle$.

Example 6.1 The following TRS $\mathcal{R}_{rev}$ specifies a program which computes the reverse of input lists:

$$\mathcal{R}_{rev} \left\{ \begin{array}{ll} \mathsf{rev}([\,]) & \to [\,] \\ \mathsf{rev}(x : xs) & \to \mathsf{app}(\mathsf{rev}(xs), x : [\,]) \\ \mathsf{app}([\,], ys) & \to ys \\ \mathsf{app}(x : xs, ys) & \to x : \mathsf{app}(xs, ys) \end{array} \right.$$

Let $\mathcal{P}_{com}$ be the the common part of $\mathcal{P}$ and $\mathcal{P}'$. The TRS pattern $\mathcal{P}$ does not match $\mathcal{R}_{rev}$ even though the TRS pattern $\mathcal{P} \setminus \mathcal{P}_{com}$ matches the first two rules of $\mathcal{R}_{rev}$. Hence, the template $\langle \mathcal{P}_1, \mathcal{P}'_1, \mathcal{H} \rangle = \langle \mathcal{P} \setminus \mathcal{P}_{com}, \mathcal{P}' \setminus \mathcal{P}_{com}, \mathcal{H} \rangle$ can be used to transform $\mathcal{R}_{rev}$ into the following $\mathcal{R}'_{rev}$:

$$\mathcal{R}'_{rev}\left\{\begin{array}{lcl}\mathsf{rev}(xs) & \to & \mathsf{rev1}(xs,[\,])\\ \mathsf{rev1}([\,],ys) & \to & ys\\ \mathsf{rev1}(x:xs,ys) & \to & \mathsf{rev1}(xs,x:ys)\\ \mathsf{app}([\,],ys) & \to & ys\\ \mathsf{app}(x:xs,ys) & \to & x:\mathsf{app}(xs,ys)\end{array}\right.$$

Remember that the templates have to be *developed* to guarantee the correctness of the transformation. In fact, the template $\langle \mathcal{P}\backslash\mathcal{P}_{com}, \mathcal{P}'\backslash\mathcal{P}_{com}, \mathcal{H}\rangle$ is not developed and it may produce incorrect transformations. The situation is that the rules of $\mathcal{P}_{com}$ are required to show that the template $\langle \mathcal{P}, \mathcal{P}', \mathcal{H}\rangle$ is developed.

How can we develop suitable transformation templates for $\mathcal{R}_{rev}$? In fact, this can be done by moving the common part $\mathcal{P}_{com}$ into a hypothesis like this:

$$\tilde{\mathcal{P}}\ \left\{\begin{array}{lcl}\mathsf{f}(\mathsf{a}) & \to & \mathsf{b}\\ \mathsf{f}(\mathsf{c}(u,v)) & \to & \mathsf{g}(\mathsf{e}(u),\mathsf{f}(v))\end{array}\right.$$

$$\tilde{\mathcal{P}}'\ \left\{\begin{array}{lcl}\mathsf{f}(u) & \to & \mathsf{f}_1(u,\mathsf{b})\\ \mathsf{f}_1(\mathsf{a},u) & \to & u\\ \mathsf{f}_1(\mathsf{c}(u,v),w) & \to & \mathsf{f}_1(v,\mathsf{g}(w,\mathsf{e}(u)))\end{array}\right.$$

$$\tilde{\mathcal{H}}\ \left\{\begin{array}{lcl}\mathsf{g}(u,\mathsf{b}) & \approx & u\\ \mathsf{g}(\mathsf{b},u) & \approx & u\\ \mathsf{g}(\mathsf{g}(u,v),w) & \approx & \mathsf{g}(u,\mathsf{g}(v,w))\end{array}\right.$$

Then, we can easily show that there exists an equivalent transformation from $\tilde{\mathcal{P}}$ to $\tilde{\mathcal{P}}'$ under $\tilde{\mathcal{H}}$ developing the template $\langle \mathcal{P}, \mathcal{P}', \mathcal{H}\rangle$ in a similar way. Thus, the template $\langle \tilde{\mathcal{P}}, \tilde{\mathcal{P}}', \tilde{\mathcal{H}}\rangle$ is developed so that the correctness of transformations by $\langle \tilde{\mathcal{P}}, \tilde{\mathcal{P}}', \tilde{\mathcal{H}}\rangle$ is guaranteed.

Using the template $\langle \tilde{\mathcal{P}}, \tilde{\mathcal{P}}', \tilde{\mathcal{H}}\rangle$, $\mathcal{R}_{sum}$ is transformed into $\mathcal{R}'_{sum}$ where $\mathcal{R}_{com} = \{+(0,y) \to y,\ +(\mathsf{s}(x),y) \to \mathsf{s}(+(x,y))\}$. Also, $\mathcal{R}_{cat}$ is transformed into $\mathcal{R}'_{cat}$ where $\mathcal{R}_{com} = \{\mathsf{app}([\,],y) \to y,\ \mathsf{app}(x{:}xs,ys) \to x{:}\mathsf{app}(xs,ys)\}$. The TRS $\mathcal{R}_{rev}$ is transformed into the following $\mathcal{R}''_{rev}$ where $\mathcal{R}_{com} = \{\mathsf{app}([\,],ys) \to ys,\ \mathsf{app}(x{:}xs,ys) \to x{:}\mathsf{app}(xs,ys)\}$:

$$\mathcal{R}''_{rev}\left\{\begin{array}{ll}\mathsf{rev}(xs) & \to \mathsf{rev1}(xs,[\,])\\ \mathsf{rev1}([\,],ys) & \to ys\\ \mathsf{rev1}(x:xs,ys) & \to \mathsf{rev1}(xs,\mathsf{app}(x:[\,],ys))\\ \mathsf{app}([\,],ys) & \to ys\\ \mathsf{app}(x:xs,ys) & \to x:\mathsf{app}(xs,ys)\end{array}\right.$$

Here, the right-hand side of the third rule of $\mathcal{R}''_{rev}$ is not a normal form. By normalizing such terms, one obtains the preferrd $\mathcal{R}'_{rev}$.

Example 6.2 The following TRS $\mathcal{R}_\forall$ specifies the program that computes a logical multiplication of all elements of an input boolean list.

$$\mathcal{R}_\forall\left\{\begin{array}{ll}\mathsf{forall}([\,]) & \to \mathsf{true}\\ \mathsf{forall}(x{:}ys) & \to \mathsf{and}(x,\mathsf{forall}(ys))\\ \mathsf{and}(\mathsf{true},\mathsf{true}) & \to \mathsf{true}\\ \mathsf{and}(\mathsf{true},\mathsf{false}) & \to \mathsf{false}\\ \mathsf{and}(\mathsf{false},\mathsf{true}) & \to \mathsf{false}\\ \mathsf{and}(\mathsf{false},\mathsf{false}) & \to \mathsf{false}\end{array}\right.$$

Using the template $\langle \tilde{\mathcal{P}}, \tilde{\mathcal{P}}', \tilde{\mathcal{H}}\rangle$, $\mathcal{R}_\forall$ is transformed to the following TRS $\mathcal{R}'_\forall$:

$$\mathcal{R}'_\forall\left\{\begin{array}{lcl}\mathsf{forall}(u) & \to & \mathsf{f1}(u,\mathsf{true})\\ \mathsf{f1}([\,],u) & \to & u\\ \mathsf{f1}(u{:}v,w) & \to & \mathsf{f1}(v,\mathsf{and}(w,u))\\ \mathsf{and}(\mathsf{true},\mathsf{true}) & \to & \mathsf{true}\\ \mathsf{and}(\mathsf{true},\mathsf{false}) & \to & \mathsf{false}\\ \mathsf{and}(\mathsf{false},\mathsf{true}) & \to & \mathsf{false}\\ \mathsf{and}(\mathsf{false},\mathsf{false}) & \to & \mathsf{false}\end{array}\right.$$

where

$$\mathcal{R}_{com} = \left\{\begin{array}{lcl}\mathsf{and}(\mathsf{true},\mathsf{true}) & \to & \mathsf{true}\\ \mathsf{and}(\mathsf{true},\mathsf{false}) & \to & \mathsf{false}\\ \mathsf{and}(\mathsf{false},\mathsf{true}) & \to & \mathsf{false}\\ \mathsf{and}(\mathsf{false},\mathsf{false}) & \to & \mathsf{false}\end{array}\right.$$

6.2 Fusion Transformation

Fusion transformation [5),19)] is one of the most recognized techniques for transforming programs.

The following template $\langle \mathcal{P}_{fus1}, \mathcal{P}'_{fus1}, \mathcal{H}_{fus1}\rangle$ describes a simple fusion transformation:

$$\mathcal{P}_{fus1}\left\{\begin{array}{lcl}\mathsf{f}(u,v,w) & \to & \mathsf{g}(\mathsf{h}(u,v),w)\\ \mathsf{g}(\mathsf{a},u) & \to & \mathsf{b}(u)\\ \mathsf{g}(\mathsf{c}(u,v),w) & \to & \mathsf{e}(u,\mathsf{g}(v,w))\\ \mathsf{h}(\mathsf{a},u) & \to & \mathsf{r}(u)\\ \mathsf{h}(\mathsf{c}(u,v),w) & \to & \mathsf{c}(\mathsf{d}(u),\mathsf{h}(v,w))\end{array}\right.$$

$$\mathcal{P}'_{fus1}\left\{\begin{array}{ll}\mathsf{f}(\mathsf{a},u,v) & \to \mathsf{g}(\mathsf{r}(u),v)\\ \mathsf{f}(\mathsf{c}(u,v),w,z) & \to \mathsf{e}(\mathsf{d}(u),\mathsf{f}(v,w,z))\\ \mathsf{g}(\mathsf{a},u) & \to \mathsf{b}(u)\\ \mathsf{g}(\mathsf{c}(u,v),w) & \to \mathsf{e}(u,\mathsf{g}(v,w))\\ \mathsf{h}(\mathsf{a},u) & \to \mathsf{r}(u)\\ \mathsf{h}(\mathsf{c}(u,v),w) & \to \mathsf{c}(\mathsf{d}(u),\mathsf{h}(v,w))\end{array}\right.$$

$$\mathcal{H}_{fus1} = \emptyset$$

Note that a hypothesis is not necessary to develop the above template.

We now demonstrate that there exists an equivalent transformation form $\mathcal{P}_{fus1}$ to $\mathcal{P}'_{fus1}$ under $\mathcal{H}_{fus1}$, i.e., $\langle \mathcal{P}_{fus1}, \mathcal{P}'_{fus1}, \mathcal{H}_{fus1}\rangle$ is developed.

(1) Let $\mathcal{P}_0 = \mathcal{P}_{fus1}$.

(2) Let $\mathcal{P}_1 = \mathcal{P}_0 \cup \{\mathsf{f}(\mathsf{a},u,v) \to \mathsf{g}(\mathsf{r}(u),v)\}$. Here, we have

$$\begin{array}{lll} \mathsf{f}(\mathsf{a},u,v) & \to_{\mathcal{P}_0} & \mathsf{g}(\mathsf{h}(\mathsf{a},u),v) \\ & \to_{\mathcal{P}_0} & \mathsf{g}(\mathsf{r}(u),v). \end{array}$$

Thus, $\mathcal{P}_0 \Rightarrow \mathcal{P}_1$ by the Addition rule.

(3) Let $\mathcal{P}_2 = \mathcal{P}_1 \cup \{\mathsf{f}(\mathsf{c}(u,v),w,z) \to \mathsf{e}(\mathsf{d}(u),\mathsf{f}(v,w,z))\}$. Here, we have

$$\begin{array}{rl} \mathsf{f}(\mathsf{c}(u,v),w,z) & \to_{\mathcal{P}_1} \mathsf{g}(\mathsf{h}(\mathsf{c}(u,v),w),z) \\ & \to_{\mathcal{P}_1} \mathsf{g}(\mathsf{c}(\mathsf{d}(u),\mathsf{h}(v,w)),z) \\ & \to_{\mathcal{P}_1} \mathsf{e}(\mathsf{d}(u),\mathsf{g}(\mathsf{h}(v,w),z)) \\ & \leftarrow_{\mathcal{P}_1} \mathsf{e}(\mathsf{d}(u),\mathsf{f}(v,w,z)). \end{array}$$

Thus, $\mathcal{P}_1 \Rightarrow \mathcal{P}_2$ by the Addition rule.

(4) Applying the Elimination rule, we obtain $\mathcal{P}'_{fus1} = \mathcal{P}_2 \setminus \{\mathsf{f}(u,v,w) \to \mathsf{g}(\mathsf{h}(u,v),w)\}$.

Thus, the template $\langle \mathcal{P}_{fus1}, \mathcal{P}'_{fus1}, \mathcal{H}_{fus1} \rangle$ is developed.

Example 6.3 The following TRS $\mathcal{R}_{lenapp}$ specifies the program that computes the sum of the lengths of two lists.

$$\mathcal{R}_{lenapp} \left\{ \begin{array}{ll} \mathsf{lenapp}(x,y) & \to \mathsf{len}(\mathsf{app}(x,y)) \\ \mathsf{len}([\,]) & \to \mathsf{0} \\ \mathsf{len}(x{:}y) & \to \mathsf{s}(\mathsf{len}(y)) \\ \mathsf{app}([\,],y) & \to y \\ \mathsf{app}(x{:}y,z) & \to x{:}\mathsf{app}(y,z) \end{array} \right.$$

Using the template $\langle \mathcal{P}_{fus1}, \mathcal{P}'_{fus1}, \mathcal{H}_{fus1} \rangle$, the TRS $\mathcal{R}_{lenapp}$ is transformed into the following TRS $\mathcal{R}'_{lenapp}$ where $\mathcal{R}_{com} = \emptyset$.

$$\mathcal{R}'_{lenapp} \left\{ \begin{array}{ll} \mathsf{lenapp}([\,],u) & \to \mathsf{len}(u) \\ \mathsf{lenapp}(u{:}v,w) & \to \mathsf{s}(\mathsf{lenapp}(v,w)) \\ \mathsf{len}([\,]) & \to \mathsf{0} \\ \mathsf{len}(u{:}v) & \to \mathsf{s}(\mathsf{len}(v)) \\ \mathsf{app}([\,],u) & \to u \\ \mathsf{app}(u{:}v,w) & \to u{:}\mathsf{app}(v,w) \end{array} \right.$$

Example 6.4 The following TRS $\mathcal{R}_{ones+}$ specifies the program that returns a list whose elements are all $\mathsf{s}(0)$ and its length is the sum of two given natural numbers.

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

Using the template $\langle \mathcal{P}_{fus1}, \mathcal{P}'_{fus1}, \mathcal{H}_{fus1} \rangle$, the TRS $\mathcal{R}_{ones+}$ is transformed into the following TRS $\mathcal{R}'_{ones+}$ where $\mathcal{R}_{com} = \emptyset$.

$$\mathcal{R}'_{ones+} \left\{ \begin{array}{ll} \mathsf{ones{+}}(\mathsf{0},u) & \to \mathsf{ones}(u) \\ \mathsf{ones{+}}(\mathsf{s}(v),w) & \to \mathsf{s}(\mathsf{0}){:}\mathsf{ones{+}}(v,w) \\ \mathsf{ones}(\mathsf{0}) & \to [\,] \\ \mathsf{ones}(\mathsf{s}(x)) & \to \mathsf{s}(\mathsf{0}){:}\mathsf{ones}(x) \\ +(\mathsf{0},u) & \to u \\ +(\mathsf{s}(v),w) & \to \mathsf{s}(+(v,w)) \end{array} \right.$$

Through the above examples, the template $\langle \mathcal{P}_{fus1}, \mathcal{P}'_{fus1}, \mathcal{H}_{fus1} \rangle$ can produce simple fusion transformations.

Let us consider another template. The following template $\langle \mathcal{P}_{fus2}, \mathcal{P}'_{fus2}, \mathcal{H}_{fus2} \rangle$ produces fusion transformations that are slightly more complicated.

$$\mathcal{P}_{fus2} \left\{ \begin{array}{ll} \mathsf{h}(u) & \to \mathsf{f}(\mathsf{g}(u)) \\ \mathsf{f}(\mathsf{a}) & \to \mathsf{b} \\ \mathsf{f}(\mathsf{c}(u,v)) & \to \mathsf{d}(\mathsf{r}(u),\mathsf{f}(v)) \\ \mathsf{g}(u) & \to \mathsf{g1}(u,\mathsf{a}) \\ \mathsf{g1}(\mathsf{b},v) & \to \mathsf{k}(v) \\ \mathsf{g1}(\mathsf{d}(u,v),w) & \to \mathsf{g1}(v,\mathsf{e}(\mathsf{p}(u),w)) \end{array} \right.$$

$$\mathcal{P}'_{fus2} \left\{ \begin{array}{ll} \mathsf{h}(u) & \to \mathsf{h1}(u,\mathsf{b}) \\ \mathsf{h1}(\mathsf{a},v) & \to \mathsf{k}(v) \\ \mathsf{h1}(\mathsf{c}(u,v),w) & \to \mathsf{h1}(v,\mathsf{e}(\mathsf{p}(\mathsf{r}(u)),w)) \\ \mathsf{f}(\mathsf{a}) & \to \mathsf{b} \\ \mathsf{f}(\mathsf{c}(u,v)) & \to \mathsf{d}(\mathsf{r}(u),\mathsf{f}(v)) \\ \mathsf{g}(u) & \to \mathsf{g1}(u,\mathsf{a}) \\ \mathsf{g1}(\mathsf{b},v) & \to \mathsf{k}(v) \\ \mathsf{g1}(\mathsf{d}(u,v),w) & \to \mathsf{g1}(v,\mathsf{e}(\mathsf{p}(u),w)) \end{array} \right.$$

$$\mathcal{H}_{fus2} \quad \{\ \mathsf{f}(\mathsf{g1}(u,v)) \approx \mathsf{g1}(\mathsf{f}(u),\mathsf{f}(v))$$

We now show that there exists an equivalent transformation form $\mathcal{P}_{fus1}$ to $\mathcal{P}'_{fus1}$ under $\mathcal{H}_{fus1}$, i.e., $\langle \mathcal{P}_{fus2}, \mathcal{P}'_{fus2}, \mathcal{H}_{fus2} \rangle$ is developed.

(1) Let $\mathcal{P}_0 = \mathcal{P}_{fus2}$.

(2) Let $\mathcal{P}_1 = \mathcal{P}_0 \cup \{\mathsf{h1}(u,v) \to \mathsf{g1}(\mathsf{f}(u),v)\}$. Here, $\mathsf{h1}$ is a fresh pattern variable. Thus, $\mathcal{P}_0 \Rightarrow \mathcal{P}_1$ by the Introduction rule.

(3) Let $\mathcal{P}_2 = \mathcal{P}_1 \cup \{\mathsf{h1}(\mathsf{a},v) \to \mathsf{k}(v)\}$. Here, we have

$$\begin{array}{rl} \mathsf{h1}(\mathsf{a},v) & \to_{\mathcal{P}_1} \mathsf{g1}(\mathsf{f}(\mathsf{a}),v) \\ & \to_{\mathcal{P}_1} \mathsf{g1}(\mathsf{b},v) \\ & \to_{\mathcal{P}_1} \mathsf{k}(v) \end{array}$$

Thus, $\mathcal{P}_1 \Rightarrow \mathcal{P}_2$ by the Addition rule.

(4) Let $\mathcal{P}_3 = \mathcal{P}_2 \cup \{\mathsf{h1}(\mathsf{c}(u,v),w) \to \mathsf{h1}(v,\mathsf{e}(\mathsf{p}(\mathsf{r}(u)),w))\}$. Here, we have

$$\begin{array}{rl} \mathsf{h1}(\mathsf{c}(u,v),w) & \to_{\mathcal{P}_2} \mathsf{g1}(\mathsf{f}(\mathsf{c}(u,v)),w) \\ & \to_{\mathcal{P}_2} \mathsf{g1}(\mathsf{d}(\mathsf{r}(u),\mathsf{f}(v)),w) \\ & \to_{\mathcal{P}_2} \mathsf{g1}(\mathsf{f}(v),\mathsf{e}(\mathsf{p}(\mathsf{r}(u)),w)) \\ & \leftarrow_{\mathcal{P}_2} \mathsf{h1}(v,\mathsf{e}(\mathsf{p}(\mathsf{r}(u)),w)) \end{array}$$

Thus, $\mathcal{P}_2 \Rightarrow \mathcal{P}_3$ by the Addition rule.

(5) Let $\mathcal{P}_4 = \mathcal{P}_3 \cup \{\mathsf{h}(u) \to \mathsf{h1}(u, \mathsf{b})\}$. Here, we have

$$\begin{aligned} \mathsf{h}(u) &\to_{\mathcal{P}_3} \mathsf{f}(\mathsf{g}(u)) \\ &\to_{\mathcal{P}_3} \mathsf{f}(\mathsf{g1}(u, \mathsf{a})) \\ &\leftrightarrow_{\mathcal{H}_{fus2}} \mathsf{g1}(\mathsf{f}(u), \mathsf{f}(a)) \\ &\to_{\mathcal{P}_3} \mathsf{g1}(\mathsf{f}(u), \mathsf{b}) \\ &\leftarrow_{\mathcal{P}_3} \mathsf{h1}(u, \mathsf{b}) \end{aligned}$$

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

(6) Finally, applying the Elimination rules twice to $\mathcal{P}_4$, we obtain $\mathcal{P}'_{fus2}$.

Thus, the template $\langle \mathcal{P}_{fus2}, \mathcal{P}'_{fus2}, \mathcal{H}_{fus2} \rangle$ is developed.

Example 6.5 The following TRS $\mathcal{R}_{mdqr}$ is transformed into the TRS $\mathcal{R}'_{mdqr}$ by the template $\langle \mathcal{P}_{fus2}, \mathcal{P}'_{fus2}, \mathcal{H}_{fus2} \rangle$ where $\mathcal{R}_{com} = \{\mathsf{double}(\mathsf{0}) \to \mathsf{0},\ \mathsf{double}(\mathsf{s}(x)) \to \mathsf{s}(\mathsf{s}(\mathsf{double}(x)))\}$.

$$\mathcal{R}_{mdqr} \left\{ \begin{array}{ll} \mathsf{mapdoubleqrev}(xs) & \\ \qquad \to \mathsf{mapdouble}(\mathsf{qrev}(xs)) & \\ \mathsf{mapdouble}([\,]) & \to [\,] \\ \mathsf{mapdouble}(x{:}xs) & \\ \qquad \to \mathsf{double}(x){:}\mathsf{mapdouble}(xs) & \\ \mathsf{double}(\mathsf{0}) & \to \mathsf{0} \\ \mathsf{double}(\mathsf{s}(x)) & \to \mathsf{s}(\mathsf{s}(\mathsf{double}(x))) \\ \mathsf{qrev}(xs) & \to \mathsf{qrev1}(xs, [\,]) \\ \mathsf{qrev1}([\,], ys) & \to ys \\ \mathsf{qrev1}(x{:}xs, ys) & \to \mathsf{qrev1}(xs, x{:}ys) \end{array} \right.$$

$$\mathcal{R}'_{mdqr} \left\{ \begin{array}{ll} \mathsf{mapdoubleqrev}(u) & \\ \qquad \to \mathsf{h1}(u, [\,]) & \\ \mathsf{h1}([\,], v) & \to v \\ \mathsf{h1}(u{:}v, w) & \to \mathsf{h1}(v, \mathsf{double}(u){:}w) \\ \mathsf{mapdouble}([\,]) & \to [\,] \\ \mathsf{mapdouble}(u{:}v) & \\ \qquad \to \mathsf{double}(u){:}\mathsf{mapdouble}(v) & \\ \mathsf{qrev}(u) & \to \mathsf{qrev1}(u, [\,]) \\ \mathsf{qrev1}([\,], v) & \to v \\ \mathsf{qrev1}(u{:}v, w) & \to \mathsf{qrev1}(v, u{:}w) \\ \mathsf{double}(\mathsf{0}) & \to \mathsf{0} \\ \mathsf{double}(\mathsf{s}(x)) & \to \mathsf{s}(\mathsf{s}(\mathsf{double}(x))) \end{array} \right.$$

Through the example above, the function mapdoubleqrev is fused by the developed template $\langle \mathcal{P}_{fus2}, \mathcal{P}'_{fus2}, \mathcal{H}_{fus2} \rangle$.

7. Conclusion

We presented a rewriting framework for transforming programs with templates in this paper. To guarantee the correctness of transformation within our framework, we introduced a notion of developed templates which are constructed via the step-by-step transformations of TRS patterns. We then showed that in any transformation of programs using the developed templates the correctness of transformation could be verified automatically.

We also described the RAPT system, which implements our framework. RAPT transforms a term rewriting system according to a specified program transformation template and automatically verifies the correctness of the transformation. Examples of the developed transformation templates and their application to the transformation of program were also given.

Another implementation of program transformation using templates is the MAG system, which is based on lambda calculus [9),15)]. The correctness of transformation in MAG system is based on Huet and Lang's framework [11)]. MAG supports transformations that include modifications of expressions and matching with the help of hypothesis; its target also includes higher-order programs. RAPT does not handle such refinements, and cannot deal with most of the transformations presented by de Moor and Sittampaltam [8),15)]. The difference between MAG and RAPT, on the other hand, lies in the approach to verifying the hypothesis. Since such hypotheses are generally different in each transformation, one needs to verify them in all transformations. MAG system users usually need to verify the hypothesis by explicit induction in every different transformation. In contrast to this, RAPT proves the hypothesis automatically without needing the help of users. To the best of our knowledge, the program-transformation systems based on templates described in the literature have rarely corporated with automated theorem-proving techniques in the verification of hypotheses. RAPT involves an interesting integration of program-transformation and automated theorem-proving techniques.

References

1) Baader, F. and Nipkow, T.: *Term Rewriting and All That*, Cambridge University Press (1998).
2) Burstall, R.M. and Darlington, J.: A transformation system for developing recursive programs, *J. ACM*, Vol.24, No.1, pp.44–67 (1977).
3) Chiba, Y. and Aoto, T.: RAPT: A program transformation system based on term rewriting, *Proc. 17th International Conference on Rewriting Techniques and Applica-*

tions, Vol.4098, *LNCS*, pp.267–276, Springer-Verlag (2006).
4) Chiba, Y., Aoto, T. and Toyama, Y.: Program transformation by templates based on term rewriting, *Proc. 7th ACM-SIGPLAN International Conference on Principles and Practice of Declarative Programming* (*PPDP 2005*), pp.59–69. ACM Press (2005).
5) Chin, W.N.: Safe fusion of functional expressions II: Further improvements. *Journal of Functional Programming*, Vol.4, No.4, pp.515–555 (1994).
6) Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Tison, S. and Tommasi, M.: *Tree Automata Techniques and Applications* (1997). http://www.grappa.univ-lille3.fr/tata
7) Curien, R., Qian, Z. and Shi, H.: Efficient second-order matching. *Proc. 7th International Conference on Rewriting Techniques and Applications*, *LNCS*, Vol.1103, pp.317–331, Springer-Verlag (1996).
8) de Moor, O. and Sittampalam, G.: Generic program transformation. *Proc. 3rd International Summer School on Advanced Functional Programming*, *LNCS*, Vol.1608, pp.116–149, Springer-Verlag (1999).
9) de Moor, O. and Sittampalam. G.: Higher-order matching for program transformation, *Theoretical Computer Science*, Vol.269, pp.135–162 (2001).
10) Hirokawa, N. and Middeldorp. A.: Tsukuba termination tool, *Proc. 14th International Conference on Rewriting Techniques and Applications*, *LNCS*, Vol.2706, pp.311–320, Springer-Verlag (2003).
11) Huet, G. and Lang, B.: Proving and applying program transformations expressed with second order patterns, *Acta Informatica*, Vol.11, pp.31–55 (1978).
12) Paige, R.: Future directions in program transformations, *ACM Computing Surveys*, Vol.28, 4es, p.170 (1996).
13) Partsch, H. and Steinbrüggen, R.: Program transformation systems, *ACM Computing Surveys*, Vol.15, No.3, pp.199–236 (1983).
14) Reddy, U.S.: Term rewriting induction, *Proc. 10th International Conference on Automated Deduction*, *LNAI*, Vol.449, pp.162–177 (1990).
15) Sittampalam, G.: *Higher-Order Matching for Program Transformation*, PhD thesis, Magdalen College (2001).
16) Terese.: *Term Rewriting Systems*, Cambridge University Press (2003).
17) Toyama, Y.: Commutativity of term rewriting systems, *The Second France-Japan Artificial Intelligence and Computer Science Symposium* (1987).
18) Toyama, Y.: How to prove equivalence of term rewriting systems without induction, *Theoretical Computer Science*, Vol.90, pp.369–390 (1991).
19) Wadler, P.: Deforestation: transforming programs to eliminate trees, *Theoretical Computer Science*, Vol.73, pp.231–248 (1990).
20) Yokoyama, T., Hu, Z. and Takeichi, M.: Deterministic second-order patterns, *Information Processing Letters*, Vol.89, No.6, pp.309–314 (2004).

(Received May 1, 2006)
(Accepted August 10, 2006)

Yuki Chiba received his M.S. from Tohoku University in 2005. He is currently a doctoral student at the same university. He is a member of JSSST.

Takahito Aoto received his M.S. and Ph.D. from the 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 at Tohoku University since 2004 as an associate professor. His current research interests are term rewriting and non-classical logics. He is a member of IPSJ, JSSST, EATCS, and ASL.

Yoshihito Toyama received his B.E. from Niigata University in 1975, and his M.E. and D.E. 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 a professor at the Research Institute of Electrical Communication (RIEC) of Tohoku University. His research interests include term rewriting systems, program theory, and automated theorem proving. He is a member of IEICE, IPSJ, JSSST, ACM, and EATCS.