Program Transformation by Templates based on Term Rewriting

Yuki Chiba
RIEC, Tohoku University
2-1-1 Katahira, Aoba-ku
Sendai, 980-8577, Japan
chiba@nue.riec.tohoku.ac.jp

Takahito Aoto
RIEC, Tohoku University
2-1-1 Katahira, Aoba-ku
Sendai, 980-8577, Japan
aoto@nue.riec.tohoku.ac.jp

Yoshihito Toyama
RIEC, Tohoku University
2-1-1 Katahira, Aoba-ku
Sendai, 980-8577, Japan
toyama@nue.riec.tohoku.ac.jp

ABSTRACT

Huet and Lang (1978) presented a framework of automated program transformation based on lambda calculus in which programs are transformed according to a given program transformation template. They introduced a second-order matching algorithm of simply-typed lambda calculus to verify whether the input program matches the template. They also showed how to validate the correctness of the program transformation using the denotational semantics.

We propose in this paper a framework of program transformation by templates based on term rewriting. In our new framework, programs are given by term rewriting systems. To automate our program transformation, we introduce a term pattern matching problem and present a sound and complete algorithm that solves this problem.

We also discuss how to validate the correctness of program transformation in our framework. We introduce a notion of developed templates and a simple method to construct such templates without explicit use of induction. We then show that in any program transformation by developed templates the correctness of the transformation can be verified automatically. In our framework the correctness of the program transformation is discussed based on the operational semantics. This is a sharp contrast to Huet and Lang's framework.

Categories and Subject Descriptors

D.3.1 [**Programming Languages**]: Formal Definitions and Theory; D.1.2 [**Programming Techniques**]: Automatic Programming; F.3.1 [**Logic and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs; F.4.2 [**Mathematical Logic and Formal Languages**]: Grammars and Other Rewriting Systems

General Terms

Algorithms, Languages, Theory, Verification

Keywords

Term Rewriting, Program Transformation, Tree Homomorphism, Inductive Theorem Proving

1. INTRODUCTION

Transforming given programs automatically to optimize the efficiency is one of the most fascinating techniques for declarative programming languages [5, 18]. 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 the transformation. In their framework, programs and program schemas are given by second-order simply-typed lambda terms. They gave a second-order matching algorithm to verify whether a template can be applied to an input program. They also showed how to validate the correctness of the program transformation using the denotational semantics.

After Huet and Lang's pioneering work, Curien et al. [4] gave an improved matching algorithm by top down matching method. Yokoyama et al. [19] presented a sufficient condition to have at most one solution and a deterministic algorithm to find such a solution. Recently, de Moor and Sittampalam [6] gave a matching algorithm that can be applied also to third-order matching problems. In all of these algorithms, programs are represented by lambda terms and higher-order substitutions are performed by β-reduction of lambda calculus. But in contrast to these successive works on the matching algorithms, the formal verification part of the correctness of the transformation has been neglected. Thus the verification of correctness of the transformation still depends on Huet and Lang's original technique based on the denotational semantics.

In this paper, we propose a framework of program transformation by templates based on term rewriting. In our new framework, contrast to the existing works mentioned above, programs and program schemas are given by term rewriting systems (TRSs, short) and TRS patterns. Thus there is little difficulty to discuss the correctness of the transformation based on the operational semantics.

We introduce a notion of term homomorphisms to describe how a TRS pattern matches to a concrete TRS. We introduce a term pattern matching problem and present a sound and complete algorithm that solves this problem. A key part of our procedure of TRS transformation by templates— TRS pattern matching problem—is solved using the term pattern matching algorithm.

To guarantee the correctness of the transformation, we introduce a notion of developed templates and a simple method to construct such templates without explicit use of induction. We then show that in any program transformation by developed templates the correctness of the 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 start with some motivating examples of TRS transformation by templates, and introduce our framework. We discuss how to validate the correctness of TRS transformation by templates in Section 4. We first describe a simple technique to prove the equality of two TRSs by 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. Section 5 is devoted to a procedure of TRS transformation by templates. We present a term pattern matching algorithm and show its correctness. In Section 6, we demonstrate how the correctness of transformation by developed templates is verified automatically. We conclude our result in Section 7.

2. PRELIMINARIES

In this section, we introduce some notions of term rewriting systems [1, 15].

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 $C(\mathscr{F},\mathscr{V})$; its subset $T(\mathscr{F} \cup \{\Box_i \mid 1 \leq i \leq n\},\mathscr{V})$ is denoted by $C_n(\mathscr{F},\mathscr{V})$. $C(\mathscr{F})$ and $C_n(\mathscr{F})$ 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 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 $\xrightarrow{*}_{\mathcal{R}}$, the transitive closure by $\xrightarrow{+}_{\mathcal{R}}$, and the equivalence closure by $\xleftrightarrow{*}_{\mathcal{R}}$. 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 $\xleftrightarrow{*}_{\mathcal{E}}$.

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. 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, $s \xrightarrow{*}_{\mathcal{R}} s_1$ and $s \xrightarrow{*}_{\mathcal{R}} s_2$ imply that there exists a term t such that $s_1 \xrightarrow{*}_{\mathcal{R}} t$ and $s_2 \xrightarrow{*}_{\mathcal{R}} t$. Note that $\mathrm{CR}(\mathcal{R})$, $s, t \in \mathrm{NF}(\mathcal{R})$ and $s \xleftrightarrow{*}_{\mathcal{R}} t$ 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

In this section, we introduce our framework of program transformation in which programs are formalized by TRSs. Let us start with some motivating examples.

Example 1. A program that computes the summation of a list is specified by the following TRS $\mathcal{R}$, in which the natural numbers $0, 1, 2, \ldots$ are expressed as $\mathsf{0}, \mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{s}(\mathsf{0})), \ldots$.

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

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

Using the well-known transformation from the recursive form to the iterative (tail-recursive) form, the following different TRS $\mathcal{R}'$ for the list summation program is obtained:

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

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

Example 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 $\mathcal{R}$.

$$\mathcal{R} \quad \begin{cases} \mathsf{cat}([\,]) & \to \quad [\,] \\ \mathsf{cat}(x_1{:}y_1) & \to \quad \mathsf{app}(x_1, \mathsf{cat}(y_1)) \\ \mathsf{app}([\,], x_2) & \to \quad x_2 \\ \mathsf{app}(x_3{:}y_3, z_3) & \to \quad x_3{:}\mathsf{app}(y_3, z_3) \end{cases}$$

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

$$\mathcal{R}' \quad \begin{cases} \mathsf{cat}(x_4) & \to \quad \mathsf{cat1}(x_4, [\,]) \\ \mathsf{cat1}([\,], x_5) & \to \quad x_5 \\ \mathsf{cat1}(x_6{:}y_6, z_6) & \to \quad \mathsf{cat1}(y_6, \mathsf{app}(z_6, x_6)) \\ \mathsf{app}([\,], x_7) & \to \quad x_7 \\ \mathsf{app}(x_8{:}y_8, z_8) & \to \quad x_8{:}\mathsf{app}(y_8, z_8) \end{cases}$$

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

$$\mathcal{P} \quad \begin{cases} \mathsf{f}(\mathsf{a}) & \to \quad \mathsf{b} \\ \mathsf{f}(\mathsf{c}(u_1, v_1)) & \to \quad \mathsf{g}(u_1, \mathsf{f}(v_1)) \\ \mathsf{g}(\mathsf{b}, u_2) & \to \quad u_2 \\ \mathsf{g}(\mathsf{d}(u_3, v_3), w_3) & \to \quad \mathsf{d}(u_3, \mathsf{g}(v_3, w_3)) \end{cases}$$

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

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

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 $\mathcal{R}$ and $\mathcal{R}'$ is guaranteed when the instantiation satisfies the following equations, called *hypothesis*:

$$\mathcal{H} \quad \begin{cases} \mathsf{g}(\mathsf{b}, u_1) & \approx \quad \mathsf{g}(u_1, \mathsf{b}) \\ \mathsf{g}(\mathsf{g}(u_2, v_2), w_2) & \approx \quad \mathsf{g}(u_2, \mathsf{g}(v_2, w_2)). \end{cases}$$

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

Definition 1. Let $\mathscr{P}$ be a set of pattern variables (disjoint from $\mathscr{F}$ and $\mathscr{V}$) where each pattern variable $p \in \mathscr{P}$ has its arity (denoted by $\mathrm{arity}(p)$). A *pattern* is a term in $T(\mathscr{F} \cup \mathscr{P}, \mathscr{V})$. 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 the program transformation by 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 [3]—we call it a *term homomorphism*.

Definition 2. Let φ be a mapping from $\mathscr{P} \cup \mathscr{V}$ to $C(\mathscr{P} \cup \mathscr{F}, \mathscr{V})$. We say φ is a *term homomorphism* if the following conditions are satisfied:

1. $\varphi(p) \in \mathrm{C}_{\mathrm{arity}(p)}(\mathscr{F})$ for any $p \in \mathrm{dom}_{\mathscr{P}}(\varphi)$,
2. $\varphi(x) \in \mathscr{V}$ for any $x \in \mathrm{dom}_{\mathscr{V}}(\varphi)$,
3. φ is injective on $\mathrm{dom}_{\mathscr{V}}(\varphi)$, that is, for any $x, y \in \mathrm{dom}_{\mathscr{V}}(\varphi)$, if $x \neq y$ then $\varphi(x) \neq \varphi(y)$,

where $\mathrm{dom}_{\mathscr{P}}(\varphi) = \{p \in \mathscr{P} \mid \varphi(p) \neq p(\Box_1, \ldots, \Box_{\mathrm{arity}(p)})\}$ 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 $T(\mathscr{F} \cup \mathscr{P}, \mathscr{V})$ as follows:

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

We 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.

The following basic property of the term homomorphisms is proved in a straightforward way using injectivity of term homomorphisms.

PROPOSITION 1 (PRESERVATION OF REDUCTION). *Let φ be a term homomorphism. If $s \to_{\mathcal{P}} t$ ($s \leftrightarrow_{\mathcal{H}} t$) then we have $\varphi(s) \to_{\varphi(\mathcal{P})} \varphi(t)$ (resp. $\varphi(s) \leftrightarrow_{\varphi(\mathcal{H})} \varphi(t)$).*

The TRS transformation by template is defined as follows.

Definition 3. 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 $\mathcal{R} = \varphi(\mathcal{P})$ and $\mathcal{R}' = \varphi(\mathcal{P}')$.

We 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 4. Let $\mathcal{R}$, $\mathcal{R}'$ be the TRSs in Example 1, and $\langle \mathcal{P}, \mathcal{P}', \mathcal{H} \rangle$ the template given in Example 3. Then the following term homomorphism φ satisfies $\mathcal{R} = \varphi(\mathcal{P})$ and $\mathcal{R}' = \varphi(\mathcal{P}')$.

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

Thus the TRS $\mathcal{R}$ is transformed into $\mathcal{R}'$ by $\langle \mathcal{P}, \mathcal{P}', \mathcal{H} \rangle$.

Example 5. Let $\mathcal{R}$, $\mathcal{R}'$ be the TRSs in Example 2, and $\langle \mathcal{P}, \mathcal{P}', \mathcal{H} \rangle$ the template given in Example 3. Then the following term homomorphism φ satisfies $\mathcal{R} = \varphi(\mathcal{P})$ and $\mathcal{R}' =$

$\varphi(\mathcal{P}')$.

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

Thus the TRS $\mathcal{R}$ is transformed into $\mathcal{R}'$ by $\langle \mathcal{P}, \mathcal{P}', \mathcal{H} \rangle$.

Readers easily observe that in these examples it is shown in the same way that $\mathcal{R}$ is transformed into $\mathcal{R}'$. From this observation a natural question arises: *does the template guarantee the correctness of all the transformations by that template?* In the next section, we will discuss a criteria of the templates for the correct transformation and give a positive answer to this question.

4. CORRECTNESS OF THE TEMPLATES

In this section, we discuss how to validate the correctness of program transformation by templates, that is, when the equivalence of the input and the output programs of the 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. Let $\mathscr{G}$ be a set of function symbols such that $\mathscr{F}_c \subseteq \mathscr{G} \subseteq \mathscr{F}$. 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 $t \in T(\mathscr{F}_c)$, $s \xrightarrow{*}_{\mathcal{R}} t$ iff $s \xrightarrow{*}_{\mathcal{R}'} t$ holds.

A short remark on the definition of the equivalence of TRSs. In a program transformation from $\mathcal{R}$ to $\mathcal{R}'$, one can not 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 $t \in T(\mathscr{F}_c)$. 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, functions in $\mathscr{G}$ are those originally requested to compute by the TRSs in comparison.

Example 6. Let us consider $\mathcal{R}$ and $\mathcal{R}'$ in Example 1. Then $\mathsf{sum1}([\,], \mathsf{s}(0)) \to_{\mathcal{R}'} \mathsf{s}(0) \in T(\mathscr{F}_c)$, but $\mathsf{sum1}([\,], \mathsf{s}(0))$ is in a normal form of $\mathcal{R}$, because $\mathcal{R}$ has no rewrite rules for $\mathsf{sum1}$. Thus $\mathcal{R} \not\simeq_{\mathscr{G}} \mathcal{R}'$ for any $\mathscr{G}$ containing $\mathsf{sum1}$. Rather, one should consider the equivalence of these TRSs by setting $\mathscr{G} = \{\mathsf{sum}, +, :, [\,], \mathsf{s}, 0\}$; indeed, in that case one can prove $\mathcal{R} \simeq_{\mathscr{G}} \mathcal{R}'$.

Although whether two TRSs are equivalent is an undecidable problem in general, for some restricted class of left-linear TRSs, it is known that two TRSs are equivalent when there exists an *equivalent transformation* from one to the other [17]. We 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 5. 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*

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

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*

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

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

(E) *Elimination*

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

When this is the case, we write $\mathcal{R}_k \Rightarrow \mathcal{R}_{k+1}$. (In 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 $\mathcal{R} \underset{I}{\overset{*}{\Rightarrow}} \cdot \underset{A}{\overset{*}{\Rightarrow}} \cdot \underset{E}{\overset{*}{\Rightarrow}} \mathcal{R}'$ under $\mathcal{E}$.

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

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 $s \xrightarrow{*}_{\mathcal{R}} t$. 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 $s \approx t \in \mathcal{E}$.

The following result is obtained by modifying Theorem 7.1 of [17].

THEOREM 1. *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 $\mathcal{R} \underset{I}{\overset{*}{\Rightarrow}} \mathcal{R}_I \underset{A}{\overset{*}{\Rightarrow}} \mathcal{R}_A \underset{E}{\overset{*}{\Rightarrow}} \mathcal{R}'$. 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 Introduction rule. Thus by our assumption $\mathrm{SC}(\mathcal{R}, \mathscr{G})$ it easily follows by induction on the length of $\mathcal{R} \underset{I}{\overset{*}{\Rightarrow}} \mathcal{R}_i$ that $\mathrm{SC}(\mathcal{R}_i, \mathscr{F}_i)$ for all i such that $\mathcal{R} \underset{I}{\overset{*}{\Rightarrow}} \mathcal{R}_i$. Thus we may assume w.l.o.g. $\mathrm{SC}(\mathcal{R}_I, \mathscr{F})$, because we may forget any function symbols not appearing even in $\mathcal{R}_I$. It is clear that $\mathcal{R} \subseteq \mathcal{R}_I$ by the definition of 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[16, 17]. 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 $\mathscr{F}$. It is easy to see by the definition of 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 is follows easily from the item 2 and the definition of 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, $s \overset{*}{\to}_{\mathcal{R}} t' = t$. $\square$

Example 7. Let $\mathcal{R}$, $\mathcal{R}'$ be the TRSs in Example 1. Let $\mathcal{E}$ be the following set of equations.

$$\mathcal{E} \quad \left\{ \begin{array}{lcl} +(0,x) & \approx & +(x,0) \\ +(+(x,y),z) & \approx & +(x,+(y,z)) \end{array} \right.$$

Note that any equation in $\mathcal{E}$ is an inductive consequence of $\mathcal{R}$ for $\mathscr{G} = \{\mathsf{sum},+,:,[\,],\mathsf{s},0\}$, i.e., $\mathcal{R},\mathscr{G} \vdash_{ind} \mathcal{E}$[1].

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

1. Let
   $$\mathcal{R}_1 \quad = \quad \mathcal{R}_0 \cup \{\mathsf{sum}_1(x,y) \to +(y,\mathsf{sum}(x))\}$$
   Clearly, $\mathcal{R}_0 \Rightarrow \mathcal{R}_1$ by the Introduction rule.
2. Let $\mathcal{R}_2 = \mathcal{R}_1 \cup \{\mathsf{sum}(x) \to \mathsf{sum}_1(x,0)\}$. Here, we have
   $$\begin{array}{lll} \mathsf{sum}(x) & \leftarrow_{\mathcal{R}_1} & +(0,\mathsf{sum}(x)) \\ & \leftarrow_{\mathcal{R}_1} & \mathsf{sum}_1(x,0) \end{array}$$
   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
   $$\begin{array}{lll} \mathsf{sum}_1([\,],x) & \to_{\mathcal{R}_2} & +(x,\mathsf{sum}([\,])) \\ & \to_{\mathcal{R}_2} & +(x,0) \\ & \leftrightarrow_{\mathcal{E}} & +(0,x) \\ & \to_{\mathcal{R}_2} & x \end{array}$$
   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 $\mathcal{R}'$.

Thus, there exists an equivalent transformation from $\mathcal{R}$ to $\mathcal{R}'$ under $\mathcal{E}$. It is easily shown that $\mathcal{R}$ is confluent and sufficient complete for $\mathscr{G}$ and that $\mathcal{R}'$ is sufficient complete for $\mathscr{G} \cup \{\mathsf{sum1}\}$. Therefore, from Theorem 1, it follows that $\mathcal{R} \simeq_{\mathscr{G}} \mathcal{R}'$.

For the TRS transformation in Example 2, it is easily observed that the correctness of the transformation is proved exactly by the same way. Thus one may naturally expect that such manual transformations can be conducted at the template level. A naive method to prove 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 $\mathcal{H}$ similarly 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 $\mathcal{P} = \{\mathsf{p}(x) \to \mathsf{a}\}$ and $\mathcal{P}' = \mathcal{P} \cup \{\mathsf{q}(x) \to \mathsf{b}\}$. Then, $\mathcal{P} \underset{I}{\Rightarrow} \mathcal{P}'$. However, $\varphi(\mathcal{P}) \underset{I}{\not\Rightarrow} \varphi(\mathcal{P}')$ when $\varphi = \{\mathsf{p} \mapsto \mathsf{f}(\square_1), \mathsf{q} \mapsto \mathsf{f}(\square_1), \mathsf{a} \mapsto 0, \mathsf{b} \mapsto 1\}$. The key idea on proof of Theorem 1 is the preservation of the Church-Rosser property under Introduction rule. In the example above, $\varphi(\mathcal{P}')$ does not have the Church-Rosser property even though $\mathcal{P}'$ has it. Thus, in order to preserve the correctness of each step, in particular *Introduction* step, in the equivalence transformation, some restriction on the term homomorphism φ is necessary. This leads us to the notion of CS homomorphisms.

[1]Precisely speaking, we should consider the many-sorted language to claim this; see Section 6 for details.

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

Definition 6. A term homomorphism φ is a *CS homomorphism* if (1) 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 by these inference rules then the correctness of all the TRS transformations obtained by this template are verified *automatically.*

Definition 7. Let $\mathcal{P}_0$ be a left-linear CS pattern over $\mathscr{P}_0$ and $\mathcal{H}$ a set of equations over $\mathscr{P}_0$. An equivalent transformation sequence under $\mathcal{H}$ is a sequence $\mathcal{P}_0, \ldots, \mathcal{P}_n$ of CS patterns (over $\mathscr{P}_0, \ldots, \mathscr{P}_n$ respectively) such that $\mathcal{P}_{k+1}$ is obtained from $\mathcal{P}_k$ by applying one of the following inference rules:

(I) *Introduction*

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

provided that $p(x_1, \ldots, x_n) \to r$ is a left-linear rewrite rule such that $p \in \mathscr{P}_{\mathrm{d}} \setminus \mathscr{P}_k$ and $r \in T(\mathscr{P}_k \cap \mathscr{P}_{\mathrm{d}}, \mathscr{V})$. We put $\mathscr{P}_{k+1} = \mathscr{P}_k \cup \{p\}$.

(A) *Addition*

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

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

(E) *Elimination*

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

(In Addition and Elimination rules, $\mathscr{P}_{k+1}$ can be any set of pattern variables such that $\mathscr{P}_{k+1} \subseteq \mathscr{P}_k$ provided that $\mathcal{P}_{k+1}$ is a TRS pattern over $\mathscr{P}_{k+1}$.) 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 $\mathcal{P} \underset{I}{\overset{*}{\Rightarrow}} \cdot \underset{A}{\overset{*}{\Rightarrow}} \cdot \underset{E}{\overset{*}{\Rightarrow}} \mathcal{P}'$ under $\mathcal{H}$.

Definition 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 9. Let $\mathscr{G}$ be a set of function symbols such that $\mathscr{F}_{\mathrm{c}} \subseteq \mathscr{G} \subseteq \mathscr{F}$ 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 $\mathcal{R} = \varphi(\mathcal{P})$, $\mathcal{R}' = \varphi(\mathcal{P}')$, and $\mathcal{R}, \mathscr{G} \vdash_{ind} \varphi(\mathcal{H})$.

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

LEMMA 1. *Let φ be a CS homomorphism. If $\mathcal{P} \underset{I}{\Rightarrow} \mathcal{P}'$ then $\varphi(\mathcal{P}) \underset{I}{\Rightarrow} \varphi(\mathcal{P}')$.*

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 Introduction rule that $\mathrm{root}(\varphi(p))$ does not occur in $\varphi(\mathcal{P})$, that $\varphi(p(x_1, \ldots, x_n))$ is linear, and that all the function symbols in $\varphi(r)$ occurs in $\varphi(\mathcal{P})$ due to $r \in T(\mathscr{P} \cap \mathscr{P}_{\mathrm{d}})$. $\Box$

LEMMA 2. *Let φ be a CS homomorphism. If $\mathcal{P} \underset{A}{\Rightarrow} \mathcal{P}'$ then $\varphi(\mathcal{P}) \underset{A}{\Rightarrow} \varphi(\mathcal{P}')$.*

PROOF. This immediately follows from Proposition 1. $\Box$

Thus, we arrive at

THEOREM 2 (CORRECTNESS OF TEMPLATE). *Let $\mathscr{G}$ and $\mathscr{G}'$ be sets of function symbols such that $\mathscr{F}_{\mathrm{c}} \subseteq \mathscr{G}, \mathscr{G}' \subseteq \mathscr{F}$. 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 1 and 2, and Theorem 1. $\Box$

Example 8. We demonstrate how to develop a template. Let $\langle \mathcal{P}, \mathcal{P}', \mathcal{H} \rangle$ be the transformation template in Example 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

$$\mathcal{P}_1 \quad = \quad \mathcal{P}_0 \cup \{\mathsf{f}_1(u, v) \to \mathsf{g}(v, \mathsf{f}(u))\}$$

   Here, f_1 is a fresh function symbol. 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

$$\begin{array}{lll} \mathsf{f}(u) & \leftarrow_{\mathcal{P}_1} & \mathsf{g}(\mathsf{b}, \mathsf{f}(u)) \\ & \leftarrow_{\mathcal{P}_1} & \mathsf{f}_1(u, \mathsf{b}). \end{array}$$

   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

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

   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

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

   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. AUTOMATING THE TRS TRANSFORMATIONS

In this section, we describe a procedure for TRS transformation by templates. We mainly focus on the TRS pattern matching problem, which is a key part of our procedure of TRS transformation by templates. We introduce a term pattern matching problem and present a sound and complete algorithm that solves this problem. Then the algorithm that solves TRS pattern matching problem is obtained using the term pattern matching algorithm.

Definition 10. (1) A pair $\langle s,t\rangle$ of a pattern s and a term t is called a *(term pattern) matching pair.* A matching pair $\langle s,t\rangle$ is written as $s \trianglelefteq t$. A *(term pattern) matching problem* is a finite set of matching pairs. (2) For a matching pair $s \trianglelefteq t$, we say s *matches* t when there exists a term homomorphism φ such that $\varphi(s) = t$; the term homomorphism φ is called a *matcher* (or *solution*) of $s \trianglelefteq t$. (3) A matching problem S is *trivial* when $s = t$ for all $s \trianglelefteq t \in S$. For a term homomorphism φ and a matching problem S, let $\varphi(S) = \{\varphi(s) \trianglelefteq t \mid s \trianglelefteq t \in S\}$. When $\varphi(S)$ is trivial, φ is said to be a matcher (or solution) of the matching problem S.

Example 9. Suppose $\mathsf{f}, \mathsf{c} \in \mathscr{P}$ and $\mathsf{sum}, : \in \mathscr{F}$. Then $\mathsf{f}(\mathsf{c}(u,v)) \trianglelefteq \mathsf{sum}(x{:}y)$ is a matching pair. Let $S = \{\ \mathsf{f}(\mathsf{c}(u,v)) \trianglelefteq \mathsf{sum}(x{:}y)\ \}$ be a matching problem and $\varphi = \{\ \mathsf{f} \mapsto \mathsf{sum}(\Box_1),\ \mathsf{c} \mapsto \Box_1{:}\Box_2,\ u \mapsto x,\ v \mapsto y\ \}$ a term homomorphism. Then $\varphi(S) = \{\mathsf{sum}(x{:}y) \trianglelefteq \mathsf{sum}(x{:}y)\}$ is a trivial matching problem and thus φ is a matcher of S. We also write $\varphi(S)$ as $\{\mathsf{f} \mapsto \mathsf{sum}(\Box_1), \mathsf{c} \mapsto \Box_1{:}\Box_2, u \mapsto x, v \mapsto y\}S$.

We next give a procedure **Match** that computes a matcher of S non-deterministically for a given matching problem S when it succeeds.

Definition 11. (1) Let $\Longrightarrow$ be a relation between pairs of a matching problem and a term homomorphism defined by: $\langle S,\varphi\rangle \Longrightarrow \langle S',\varphi'\rangle$ when $\langle S,\varphi\rangle$ is rewritten to $\langle S',\varphi'\rangle$ by an application of the rules **Bound**, **Split** or **Extract** in Table 1. Let $\stackrel{*}{\Longrightarrow}$ be the reflexive transitive closure of $\Longrightarrow$. (2) The procedure **Match** is given as follows:

Match

Input: a matching problem S
Output: a term homomorphism φ

1. Repeatedly apply inference rules **Bound**, **Split** or **Extract** starting from $\langle S,\emptyset\rangle$.

2. Output φ if $\langle S,\emptyset\rangle \stackrel{*}{\Longrightarrow} \langle\emptyset,\varphi\rangle$.

Example 10. We demonstrate a sequence $\langle S,\emptyset\rangle \stackrel{*}{\Longrightarrow} \langle\emptyset,\varphi\rangle$ in the procedure **Match** for an input $S = \{\ \mathsf{f}(\mathsf{c}(u,v)) \trianglelefteq \mathsf{sum}(x{:}y),\ \mathsf{g}(u,\mathsf{f}(v)) \trianglelefteq +(x,\mathsf{sum}(y))\ \}$ in Figure 1.

The next lemma is readily checked.

LEMMA 3 (**Match** IS WELL-DEFINED). *By applying inference rules **Bound**, **Split** or **Extract** any pair of a matching problem and a term homomorphism is rewritten to a pair of a matching problem and a term homomorphism, that is, the procedure **Match** is well-defined.*

Our next aim in this section is to prove the correctness of the procedure **Match**. First, we show that for a given input the procedure **Match** terminates and that the set of all possible outputs of the algorithm is finite. These facts are necessary to show the soundness and completeness of the procedure **Match**.

THEOREM 3 (TERMINATION OF **Match**). *The procedure **Match** terminates for any input.*

PROOF. Clearly, it suffices to show that $\Longrightarrow$ is noetherian. Let $>$ be the usual order on the set of positive natural numbers, $>\times>$ the lexicographic extension of $>$ from left to right, and $>_m$ the multiset extension of $>\times>$. Moreover, let $\gg$ be a partial order on the set of pairs of a matching problem and a term homomorphism given by: $\langle S,\varphi\rangle \gg \langle S',\varphi'\rangle$ iff $[\langle|\mathscr{P}(s)|,|s|\rangle \mid s \trianglelefteq t \in S] >_m [\langle|\mathscr{P}(s)|,|s|\rangle \mid s \trianglelefteq t \in S']$ where $|\mathscr{P}(s)|$ denotes the cardinality of the set of pattern variables appear in s and $|s|$ denotes the term size of s. Since the order $>_m$ is well-founded, so is the order $\gg$. Then for each of the inference rules, it is easy to show $\langle S,\varphi\rangle \Longrightarrow \langle S',\varphi'\rangle$ implies $\langle S,\varphi\rangle \gg \langle S',\varphi'\rangle$. Hence $\Longrightarrow$ is noetherian. $\Box$

We now know the procedure **Match** terminates and therefore use the term *algorithm* instead of the procedure.

THEOREM 4 (NUMBER OF OUTPUTS). *For any given input, the number of outputs of the algorithm **Match** is finite.*

PROOF. Clearly, the number of non-deterministic choices of the procedure **Match** is finite. Thus, because the procedure **Match** is terminating, the number of possible outputs of the algorithm **Match** is finite. $\Box$

We next give proofs of the soundness and the completeness of the algorithm **Match**. These are proved by induction on the length of the sequence $\langle S,\emptyset\rangle \stackrel{*}{\Longrightarrow} \langle\emptyset,\varphi\rangle$. For this, it is convenient to have a notion of a solution for a pair $\langle S,\varphi\rangle$ of a matching problem S and a term homomorphism φ.

Definition 12. Let S be a matching problem and φ a term homomorphism. A term homomorphism $\tilde{\varphi}$ is said to be a *solution* of the pair $\langle S,\varphi\rangle$ if (1) $\tilde{\varphi}(S)$ is trivial and (2) $\varphi \subseteq \tilde{\varphi}$.

LEMMA 4. *Let S, S' be matching problems and $\varphi, \varphi', \tilde{\varphi}$ term homomorphisms. Suppose $\langle S,\varphi\rangle \Longrightarrow \langle S',\varphi'\rangle$ and $\tilde{\varphi}$ is a solution of $\langle S',\varphi'\rangle$. Then $\tilde{\varphi}$ is a solution of $\langle S,\varphi\rangle$.*

PROOF. Distinguish cases by the inference rule applied in the step $\langle S,\varphi\rangle \Longrightarrow \langle S',\varphi'\rangle$. $\Box$

THEOREM 5 (SOUNDNESS OF **Match**). *Let S be a matching problem and φ an output of the algorithm **Match** for the input S. Then φ is a matcher of S.*

PROOF. Using Lemma 4, it is easy to show by induction on the length of $\langle S',\varphi'\rangle \stackrel{*}{\Longrightarrow} \langle S'',\varphi''\rangle$ that if $\tilde{\varphi}$ is a solution of $\langle S'',\varphi''\rangle$ then it is also a solution of $\langle S',\varphi'\rangle$. The claim follows immediately from this. $\Box$

We next show the completeness of the algorithm **Match**. To state the completeness in a precise way, we introduce the notion of a complete set of matchers.

Definition 13. Let S be a matching problem and Φ a set of term homomorphisms. The set Φ is said to be a *complete set of matchers of* S when the following conditions are satisfied: (1) any term homomorphism $\varphi \in \Phi$ is a matcher of S; (2) for any matcher φ' of S, there exists $\varphi \in \Phi$ such that $\varphi \subseteq \varphi'$.

1. **Bound**

$$\frac{\langle S \cup \{x \trianglelefteq y\},\ \varphi\rangle}{\langle S,\ \varphi \cup \{x \mapsto y\}\rangle} \qquad x, y \in \mathscr{V}, \varphi(x) = y \vee (x \notin \mathrm{dom}_{\mathscr{V}}(\varphi) \wedge y \notin \mathrm{range}(\varphi))$$

2. **Split**

$$\frac{\langle S \cup \{f(s_1, \ldots, s_n) \trianglelefteq f(t_1, \ldots, t_n)\},\ \varphi\rangle}{\langle S \cup \{s_1 \trianglelefteq t_1, \ldots, s_n \trianglelefteq, t_n\},\ \varphi\rangle} \qquad f \in \mathscr{F}$$

3. **Extract**

$$\frac{\langle S \cup \{p(s_1, \ldots, s_n) \trianglelefteq C\langle t_1, \ldots, t_n\rangle\},\ \varphi\rangle}{\langle \{p \mapsto C\}(S \cup \{s_i \trianglelefteq t_i \mid \Box_i \in C\}),\ \varphi \cup \{p \mapsto C\}\rangle} \quad \begin{array}{l} p \in \mathscr{P}, C \in C_n(\mathscr{F}), \\ \forall i \le n.\ \forall x \mapsto y \in \varphi.\ (\Box_i \in C \wedge y \in \mathscr{V}(t_i) \Rightarrow x \in \mathscr{V}(s_i)) \end{array}$$

Table 1: Inference rules of Match

$$\begin{array}{lll}
 & \langle \{\mathsf{f}(\mathsf{c}(u,v)) \trianglelefteq \mathsf{sum}(x{:}y),\ \mathsf{g}(u,\mathsf{f}(v)) \trianglelefteq +(x,\mathsf{sum}(y))\},\ \emptyset\rangle & = \langle S_0, \varphi_0\rangle \\
\underset{\text{Extract}}{\Longrightarrow} & \langle \{\mathsf{c}(u,v) \trianglelefteq x{:}y,\ \mathsf{g}(u,\mathsf{sum}(v)) \trianglelefteq +(x,\mathsf{sum}(y))\},\ \varphi_0 \cup \{\mathsf{f} \mapsto \mathsf{sum}(\Box_1)\}\rangle & = \langle S_1, \varphi_1\rangle \\
\underset{\text{Extract}}{\Longrightarrow} & \langle \{u \trianglelefteq x,\ v \trianglelefteq y,\ \mathsf{g}(u,\mathsf{sum}(v)) \trianglelefteq +(x,\mathsf{sum}(y))\},\ \varphi_1 \cup \{\mathsf{c} \mapsto \Box_1{:}\Box_2\}\rangle & = \langle S_2, \varphi_2\rangle \\
\underset{\text{Bound}}{\Longrightarrow} & \langle \{v \trianglelefteq y,\ \mathsf{g}(u,\mathsf{sum}(v)) \trianglelefteq +(x,\mathsf{sum}(y))\},\ \varphi_2 \cup \{u \mapsto x\}\rangle & = \langle S_3, \varphi_3\rangle \\
\underset{\text{Bound}}{\Longrightarrow} & \langle \{\mathsf{g}(u,\mathsf{sum}(v)) \trianglelefteq +(x,\mathsf{sum}(y))\},\ \varphi_3 \cup \{v \mapsto y\}\rangle & = \langle S_4, \varphi_4\rangle \\
\underset{\text{Extract}}{\Longrightarrow} & \langle \{u \trianglelefteq x,\ \mathsf{sum}(v) \trianglelefteq \mathsf{sum}(y)\},\ \varphi_4 \cup \{\mathsf{g} \mapsto +(\Box_1,\Box_2)\}\rangle & = \langle S_5, \varphi_5\rangle \\
\underset{\text{Bound}}{\Longrightarrow} & \langle \{\mathsf{sum}(v) \trianglelefteq \mathsf{sum}(y)\},\ \varphi_5 \cup \{u \mapsto x\}\rangle & = \langle S_6, \varphi_6\rangle \\
\underset{\text{Split}}{\Longrightarrow} & \langle \{v \trianglelefteq y\},\ \varphi_6\rangle & = \langle S_7, \varphi_7\rangle \\
\underset{\text{Bound}}{\Longrightarrow} & \langle \emptyset,\ \varphi_7 \cup \{v \mapsto y\}\rangle & = \langle S_8, \varphi_8\rangle
\end{array}$$

Figure 1: A sequence $\langle S, \emptyset\rangle \overset{*}{\Longrightarrow} \langle \emptyset, \varphi\rangle$ in the procedure Match

LEMMA 5. *Let $\langle S, \varphi\rangle$ be a pair of a non-empty matching problem S and a term homomorphism φ, and $\tilde{\varphi}$ its solution. Then there exists a pair $\langle S', \varphi'\rangle$ of a matching problem S' and a term homomorphism φ' such that $\tilde{\varphi}$ is a solution of $\langle S', \varphi'\rangle$ and $\langle S, \varphi\rangle \Longrightarrow \langle S', \varphi'\rangle$.*

PROOF. Let $S = S'' \cup \{s \trianglelefteq t\}$. By our assumption that $\tilde{\varphi}$ is a solution of $\langle S, \varphi\rangle$, it follows that (1) $\tilde{\varphi}(S'')$ is trivial, and (2) $\varphi \subseteq \tilde{\varphi}$. The proof proceeds by induction on the structure of s.

1. $s = x \in \mathscr{V}$.

   Then $t = \tilde{\varphi}(x)$, so let $y = \tilde{\varphi}(x)$. Then by $\varphi \subseteq \tilde{\varphi}$ and $\tilde{\varphi}(x) = y$, we have either $\varphi(x) = y$ or $x \notin \mathrm{dom}_{\mathscr{V}}(\varphi)$ and $y \notin \mathrm{range}(\varphi)$ as $\tilde{\varphi}$ is injective on $\mathrm{dom}_{\mathscr{V}}(\tilde{\varphi})$. Thus, one can apply inference rule **Bound**, and we have $\langle S, \varphi\rangle \Longrightarrow \langle S'', \varphi \cup \{x \mapsto y\}\rangle$. Then, clearly, we have $\varphi' = \varphi \cup \{x \mapsto y\} \subseteq \tilde{\varphi}$. Together with $S'' \subseteq S$, we know $\tilde{\varphi}$ is a solution of $\langle S'', \varphi'\rangle$.

2. $s = f(s_1, \ldots, s_n)$ with $f \in \mathscr{F}$.

   Then $t = \tilde{\varphi}(f(s_1, \ldots, s_n)) = f(\tilde{\varphi}(s_1), \ldots, \tilde{\varphi}(s_n))$, so let $t = f(t_1, \ldots, t_n)$ where $t_i = \tilde{\varphi}(s_i)$ for $i = 1, \ldots, n$. Then, one can apply the inference rule **Split**, and we have $\langle S, \varphi\rangle \Longrightarrow \langle S'' \cup \{s_1 \trianglelefteq t_1, \ldots, s_n \trianglelefteq t_n\}, \varphi\rangle$. It is easy to check $\tilde{\varphi}$ is a solution of $\langle S'' \cup \{s_1 \trianglelefteq t_1, \ldots, s_n \trianglelefteq t_n\}, \varphi\rangle$.

3. $s = p(s_1, \ldots, s_n)$ with $p \in \mathscr{P}$.

   Since $\tilde{\varphi}(p(s_1, \ldots, s_n)) = t$ and $t \in T(\mathscr{F}, \mathscr{V})$, $p \in \mathrm{dom}_{\mathscr{P}}(\tilde{\varphi})$. By the definition of term homomorphism, we have $\tilde{\varphi}(p) \in C_n(\mathscr{F})$. Then $t = \tilde{\varphi}(p(s_1, \ldots, s_n)) = \tilde{\varphi}(p)\langle \tilde{\varphi}(s_1), \ldots, \tilde{\varphi}(s_n)\rangle$. Let $C = \tilde{\varphi}(p)$ and $t_i = \tilde{\varphi}(s_i)$ (for $i = 1, \ldots, n$). Then $C \in C_n(\mathscr{F})$. Suppose $x \mapsto y \in \varphi$, $\Box_i \in C$, and $y \in \mathscr{V}(t_i)$. Then $x \mapsto y \in \tilde{\varphi}$ and $y \in \mathscr{V}(\tilde{\varphi}(s_i))$. Thus, since $\tilde{\varphi}$ is injective on $\mathrm{dom}_{\mathscr{V}}(\tilde{\varphi})$, it follows $x \in \mathscr{V}(s_i)$. Thus, one can apply inference rule **Extract**, and we have $\langle S, \varphi\rangle \Longrightarrow \langle \{p \mapsto C\}(S'' \cup \{s_i \trianglelefteq t_i \mid \Box_i \in C\}), \varphi \cup \{p \mapsto C\}\rangle$. Since $\{p \mapsto C\} \in \tilde{\varphi}$, we have $\tilde{\varphi}(\{p \mapsto C\}(S'' \cup \{s_i \trianglelefteq t_i \mid \Box_i \in C\})) = \tilde{\varphi}(S'' \cup \{s_i \trianglelefteq t_i \mid \Box_i \in C\})$. Thus, it is easy to check $\tilde{\varphi}$ is a solution of $\langle \{p \mapsto C\}(S'' \cup \{s_i \trianglelefteq t_i \mid \Box_i \in C\}), \varphi \cup \{p \mapsto C\}\rangle$.

□

THEOREM 6 (COMPLETENESS OF **Match**). *Let Φ be the collection of all outputs of the algorithm **Match** for the input S. Then Φ is a complete set of matchers of S.*

PROOF. By Theorem 5, any $\varphi \in \Phi$ is a matcher of S. Let $\tilde{\varphi}$ be a matcher of S. From Lemma 5, there exists a sequence $\langle S, \emptyset\rangle = \langle S_0, \varphi_0\rangle \Longrightarrow \langle S_1, \varphi_1\rangle \Longrightarrow \cdots$ of pairs of a matching problem and a term homomorphism such that $\tilde{\varphi}$ is a solution of $\langle S_i, \varphi_i\rangle$ (for $i \geq 0$). By Theorem 3, this sequence is finite. So there exists φ' such that $\langle S, \emptyset\rangle \overset{*}{\Longrightarrow} \langle \emptyset, \varphi'\rangle$ and $\varphi' \subseteq \tilde{\varphi}$. Since $\langle S, \emptyset\rangle \overset{*}{\Longrightarrow} \langle \emptyset, \varphi'\rangle$ means $\varphi' \in \Phi$, the claim follows. □

We now introduce the TRS pattern matching problem in a way similar to the term matching problem. From here on, we assume that for any TRS $\mathcal{R}$ and any defined symbol $f \in \mathscr{F}_{\mathrm{d}}$, there exists a rewrite rule $l \to r \in \mathcal{R}$ such that $\mathrm{root}(l) = f$.

Definition 14. A pair $\langle \mathcal{P}, \mathcal{R} \rangle$ of a TRS pattern $\mathcal{P}$ and TRS $\mathcal{R}$ is called a *TRS pattern matching problem.* A TRS pattern matching problem $\langle \mathcal{P}, \mathcal{R} \rangle$ is written as $\mathcal{P} \trianglelefteq \mathcal{R}$. (2) For a TRS pattern matching problem $\mathcal{P} \trianglelefteq \mathcal{R}$ we say $\mathcal{P}$ *matches* $\mathcal{R}$ when there exists a CS homomorphism φ such that $\varphi(\mathcal{P}) = \mathcal{R}$; the CS homomorphism φ is called a *matcher* (or *solution*) of $\mathcal{P} \trianglelefteq \mathcal{R}$.

By encoding $\mathcal{P}$ and $\mathcal{R}$ by sequences of patterns and terms and then running the term pattern matching algorithm, one can find a solution of the TRS pattern matching problem. Let us first demonstrate this by an example.

Example 11. Let $\mathcal{P} \trianglelefteq \mathcal{R}$ be a TRS pattern matching problem where

$$\mathcal{P} \left\{ \begin{array}{lll} \mathsf{f}(\mathsf{a}) & \to & \mathsf{b} \\ \mathsf{f}(\mathsf{c}(u_1, v_1)) & \to & \mathsf{g}(u_1, \mathsf{f}(v_1)) \end{array} \right.$$

$$\mathcal{R} \left\{ \begin{array}{lll} \mathsf{sum}([\,]) & \to & 0 \\ \mathsf{sum}(x_1{:}y_1) & \to & +(x_1, \mathsf{sum}(y_1)) \end{array} \right.$$

This TRS pattern matching problem is encoded as a term pattern patching problem

$$\begin{array}{rcl} S & = & \{\, \mathsf{f}(\mathsf{a}) \trianglelefteq \mathsf{sum}([\,]), \mathsf{b} \trianglelefteq 0, \mathsf{f}(\mathsf{c}(u_1, v_1)) \trianglelefteq \mathsf{sum}(x_1{:}y_1), \\ & & \mathsf{g}(u_1, \mathsf{f}(v_1)) \trianglelefteq +(x_1, \mathsf{sum}(y_1)) \,\}. \end{array}$$

(There are choices on which correspondence of the rules in $\mathcal{P}$ and $\mathcal{R}$ is to be chosen, but we assume that suitable such a choice has been selected in an adequate way.)

By applying the algorithm **Match** to the term pattern matching problem S, we obtain the following three solutions:

$$\left\{ \begin{array}{lll} \mathsf{f} \mapsto \mathsf{sum}(\Box_1), & \mathsf{g} \mapsto +(\Box_1, \Box_2), & \\ \mathsf{a} \mapsto [\,], & \mathsf{b} \mapsto 0, & \mathsf{c} \mapsto \Box_1{:}\Box_2, \\ u_1 \mapsto x_1, & v_1 \mapsto y_1 & \end{array} \right\},$$

$$\left\{ \begin{array}{lll} \mathsf{f} \mapsto \Box_1, & \mathsf{g} \mapsto +(\Box_2, \mathsf{sum}(\Box_1)), & \\ \mathsf{a} \mapsto \mathsf{sum}([\,]), & \mathsf{b} \mapsto 0, & \mathsf{c} \mapsto \mathsf{sum}(\Box_2{:}\Box_1), \\ u_1 \mapsto y_1, & v_1 \mapsto x_1 & \end{array} \right\},$$

$$\left\{ \begin{array}{lll} \mathsf{f} \mapsto \Box_1, & \mathsf{g} \mapsto +(\Box_1, \mathsf{sum}(\Box_2)), & \\ \mathsf{a} \mapsto \mathsf{sum}([\,]), & \mathsf{b} \mapsto 0, & \mathsf{c} \mapsto \mathsf{sum}(\Box_1{:}\Box_2), \\ u_1 \mapsto x_1, & v_1 \mapsto y_1 & \end{array} \right\}.$$

Among these solutions, one can select a CS homomorphism φ, for which $\varphi(\mathcal{P}) = \mathcal{R}$ holds. Indeed, the first term homomorphism is a CS homomorphisms.

More formally, the TRS pattern matching procedure is introduced as follows.

Definition 15. (1) Let $\mathcal{P}$ be a TRS pattern and $\mathcal{R}$ a TRS. A *sequentialization* of a TRS pattern matching problem $\mathcal{P} \trianglelefteq \mathcal{R}$ is a term pattern matching problem

$$\bigcup_{\substack{s \to t \in \mathcal{P} \\ \sigma(s \to t) = l \to r \in \mathcal{R}}} \{s \trianglelefteq l, t \trianglelefteq r\},$$

where σ maps each $s \to t \in \mathcal{P}$ to some $l \to r \in \mathcal{R}$. Note that variables of each rewriting rule are w.l.o.g. assumed to be disjoint. (2) The procedure **TRSMatch** is given like this:

TRSMatch

Input: a TRS pattern matching problem $\mathcal{P} \trianglelefteq \mathcal{R}$
Output: a CS homomorphism φ

1. Take any sequentialization of $\mathcal{P} \trianglelefteq \mathcal{R}$, and computes term homomorphism φ using **Match**.

2. output φ if φ is a CS homomorphism.

The following results follow immediately from those for the **Match**.

Theorem 7 (properties of **TRSMatch**). *For any input, the procedure **TRSMatch** terminates and the number of outputs of the algorithm **TRSMatch** is finite.*

The following theorem guarantees that any TRS pattern matching problem is solved by our TRS pattern matching algorithm.

Theorem 8 (solution of TRS pattern matching). *Let Φ be the collection of all outputs of the algorithm **TRSMatch** for the input $\mathcal{P} \trianglelefteq \mathcal{R}$. Then (1) any $\varphi \in \Phi$ is a CS homomorphism such that $\varphi(\mathcal{P}) = \mathcal{R}$; (2) if a CS homomorphism φ is a solution of the TRS pattern matching problem $\mathcal{P} \trianglelefteq \mathcal{R}$, then there exists $\tilde{\varphi} \in \Phi$ such that $\tilde{\varphi} \subseteq \varphi$.*

Proof. (1) It follows easily from the definition of **TRSMatch** and Theorem 5. (2) Suppose that a CS homomorphism φ is a solution of the TRS pattern matching problem $\mathcal{P} \trianglelefteq \mathcal{R}$. Then clearly φ is a solution of some sequentialization S of $\mathcal{P} \trianglelefteq \mathcal{R}$. Let Φ' be the set of solutions of the term pattern matching problem S. Then by Theorem 6, there exists a term homomorphism $\tilde{\varphi} \in \Phi'$ such that $\tilde{\varphi} \subseteq \varphi$ and $\tilde{\varphi}(S)$ is trivial. Thus for any $p, q \in \mathscr{P}_{\mathrm{d}}$, $\mathrm{root}(\tilde{\varphi}(p)) = \mathrm{root}(\tilde{\varphi}(q))$ implies $p = q$; for otherwise, φ is not a CS homomorphism by $\tilde{\varphi} \subseteq \varphi$. Since any defined pattern variable $p \in \mathscr{P}_{\mathrm{d}}$ appears at the root of some rewrite rule in $\mathcal{P}$, $p \in \mathrm{dom}_{\mathscr{P}}(\tilde{\varphi})$ holds. Thus for any $p \in \mathscr{P}_{\mathrm{d}}$, $\tilde{\varphi}(p) = \varphi(p) = f(\Box_{i_1}, \ldots, \Box_{i_n})$ for some $f \in \mathscr{F}_{\mathrm{d}}$. Therefore $\tilde{\varphi}$ is a CS homomorphism and hence $\tilde{\varphi} \in \Phi$. $\Box$

Finally, the procedure for TRS transformation by templates is completed like this:

TRS transformation procedure

Input: TRS $\mathcal{R}$, transformation template $\langle \mathcal{P}, \mathcal{P}', \mathcal{H} \rangle$
Output: TRS $\mathcal{R}'$

1. Using **TRSMatch**, find a CS homomorphism φ such that $\varphi(\mathcal{P}) = \mathcal{R}$.

2. If $p \in \mathscr{P} \backslash \mathrm{dom}_{\mathscr{P}}(\varphi)$ appears in $\mathcal{P}'$, then set $\varphi(p) = f(\Box_1, \ldots, \Box_{\mathrm{arity}(p)})$ for a fresh function symbol f.

3. Let $\mathcal{R}' = \varphi(\mathcal{P}')$.

Example 12. Let $\mathcal{R}$ be a TRS in Example 1 and $\langle \mathcal{P}, \mathcal{P}', \mathcal{H} \rangle$ be a template in Example 3. Below we demonstrate our TRS transformation procedure for the inputs $\mathcal{R}$ and $\langle \mathcal{P}, \mathcal{P}', \mathcal{H} \rangle$.

1. First, running the **TRSMatch** for inputs $\mathcal{P} \trianglelefteq \mathcal{R}$, the following CS homomorphism φ is found.

$$\varphi = \left\{ \begin{array}{ll} \mathsf{f} \mapsto \mathsf{sum}(\Box_1), & u_1 \mapsto x_1, \\ \mathsf{g} \mapsto +(\Box_1, \Box_2), & v_1 \mapsto y_1, \\ \mathsf{a} \mapsto [\,], & u_2 \mapsto x_2, \\ \mathsf{b} \mapsto 0, & v_3 \mapsto x_3, \\ \mathsf{c} \mapsto \Box_1{:}\Box_2, & w_3 \mapsto y_3, \\ \mathsf{d} \mapsto \mathsf{s}(\Box_2) & \end{array} \right\}$$

2. Since a pattern variable f_1 appearing in $\mathcal{P}'$ does not appear in $\mathrm{dom}(\varphi)$, we set $\varphi(\mathsf{f}_1) = \mathsf{sum1}(\Box_1, \Box_2)$. where $\mathsf{sum1}$ is a fresh function symbol.

3. Apply φ to $\mathcal{P}'$ and obtain

$$\mathcal{R}' \quad \left\{ \begin{array}{lcl} \mathsf{sum}(u_4) & \to & \mathsf{sum1}(u_4, 0) \\ \mathsf{sum1}([\,], u_5) & \to & u_5 \\ \mathsf{sum1}(u_6{:}v_6, w_6) & \to & \mathsf{sum1}(v_6, +(w_6, u_6)) \\ +(0, u_7) & \to & u_7 \\ +(\mathsf{s}(u_8), v_8) & \to & \mathsf{s}(+(u_8, v_8)) \end{array} \right.$$

Thus, the output TRS is $\mathcal{R}'$.

Our TRS matching algorithm, in particular, term pattern matching algorithm and the second-order matching algorithm in lambda calculus by Huet and Lang [4, 10, 11] seem to have an obvious resemblance although they are incomparable. In the rest of this section, we explain this briefly.

In the framework based on the lambda calculus, each program is given by a recursive program schema [15] like this:

$$\left\{ \begin{array}{lcl} \mathsf{rev}(x) & \to & \mathsf{if}(\mathsf{null}(x), [\,], \\ & & \quad \mathsf{app}(\mathsf{rev}(\mathsf{cdr}(x)), \mathsf{car}(x){:}[\,])). \end{array} \right.$$

Such a recursive program schema is represented by a lambda term using a *fixpoint operator* Y:

$$\mathsf{Y}(\lambda rev.\lambda x.\mathsf{if}(\mathsf{null}(x), [\,], \mathsf{app}(rev(\mathsf{cdr}(x)), \mathsf{car}(x){:}[\,]))),$$

or more precisely,

$$\mathsf{Y}(\lambda rev.\lambda x.\mathsf{if}\,(\mathsf{null}\,x)\,[\,]\,((\mathsf{app}\,(rev\,(\mathsf{cdr}\,x)))\,(:\,(\mathsf{car}\,x)\,[\,]))).$$

Note that the function symbol rev in the recursive program schema is changed into a (bound) variable rev in the corresponding lambda term.

On the other hand, programs represented by TRSs are not necessarily recursive program schemas. For example the similar reverse program is represented by the following TRS.

$$\left\{ \begin{array}{lcl} \mathsf{rev}([\,]) & \to & [\,] \\ \mathsf{rev}(x{:}y) & \to & \mathsf{app}(\mathsf{rev}(y), x{:}[\,]). \end{array} \right.$$

Like this, TRS may not be a recursive program schema in general. Because of this, the second-order matching algorithms in lambda calculus can not be directly applied to the TRS pattern matching problem.

6. AUTOMATED VERIFICATION OF CORRECTNESS

In this section, we give some remarks on how to automate the verification of criterion of Theorem 2 in a more realistic situation.

We take the program transformation of left-linear CS in Example 1 and the developed template of Example 3 for illustration. As a model of realistic programs, we consider many-sorted version of the TRSs and the template. All of our results in the previous sections can be easily adapted in the many-sorted framework. In our example, we consider the basic sorts $\mathrm{Nat}, \mathrm{List}$ and sorted function symbols $[\,]^{\mathrm{List}}$, $:^{\mathrm{Nat} \times \mathrm{List} \to \mathrm{List}}$, $\mathsf{sum}^{\mathrm{List} \to \mathrm{List}}$, $\mathsf{s}^{\mathrm{Nat} \to \mathrm{Nat}}$, etc.

$\mathrm{CR}(\mathcal{R})$ is verified by using the well-known criterion: (1) $\mathcal{R}$ is left-linear and non-overlapping. (2) $\mathrm{SN}(\mathcal{R})$ and all critical pairs of $\mathcal{R}$ are joinable. Criteria (1) is easily checked automatically and for usual programs the criteria (1) often suffices. For criteria (2), once $\mathrm{SN}(\mathcal{R})$ is verified then joinability of critical pairs of $\mathcal{R}$ is easily checked. $\mathrm{SN}(\mathcal{R})$ can be checked by means of termination proving techniques for TRSs, e.g. the polynomial interpretations, the simplification orderings, and/or the dependency pair method. For our example, since $\mathcal{R}$ is left-linear and non-overlapping, $\mathrm{CR}(\mathcal{R})$ holds by the criteria (1). But, for the later use, we remark that $\mathrm{SN}(\mathcal{R})$ can be proved easily by the lexicographic path ordering [12] which is builded in to most of automated termination tools for term rewriting [7, 9].

We next describe for $\mathscr{G} = \{\mathsf{sum}, :, [\,], \mathsf{s}, 0\}$ how $\mathrm{SC}(\mathcal{R}, \mathscr{G})$ is verified. For this, we assume that existence of coversets (of substitution) for sorted variables. In our example, a coverset $\Theta(x^{\mathrm{Nat}})$ for the variable x^{Nat} is given by

$$\Theta(x^{\mathrm{Nat}}) = \{\{x^{\mathrm{Nat}} \mapsto 0\}, \{x^{\mathrm{Nat}} \mapsto \mathsf{s}(y^{\mathrm{Nat}})\}\}$$

and those for the variable xs^{List} is given by

$$\Theta(xs^{\mathrm{List}}) = \{\{xs^{\mathrm{List}} \mapsto [\,]\}, \{xs^{\mathrm{List}} \mapsto y^{\mathrm{Nat}}{:}ys^{\mathrm{List}}\}\}.$$

For a many-sorted term s with $\mathscr{V}(s) = \{x_1, \ldots, x_n\}$, $s\Theta$ is given by $\{\mathrm{REN}(s\theta_1 \cdots \theta_n) \mid \theta_i \in \Theta(x_i) \text{ for } i = 1, \ldots, n\}$ where REN renames all variables distinct. For example,

$$+(x^{\mathrm{Nat}}, y^{\mathrm{Nat}})\Theta = \left\{ \begin{array}{l} +(0, 0), \\ +(0, \mathsf{s}(y^{\mathrm{Nat}}), \\ +(\mathsf{s}(x^{\mathrm{Nat}}), 0) \\ +(\mathsf{s}(x^{\mathrm{Nat}}), \mathsf{s}(y^{\mathrm{Nat}})) \end{array} \right\}.$$

For a set S of terms, $S\Theta = \bigcup\{s\Theta \mid s \in S\}$. A many-sorted TRS $\mathcal{R}$ is said to be *quasi-reducible* for $\mathscr{G}$ ($\mathrm{QR}(\mathcal{R}, \mathscr{G})$) if all elements of $\{\, f(x_1, \ldots, x_n) \mid f \in \mathscr{F}_{\mathrm{d}} \cap \mathscr{G} \,\}\Theta$ are reducible by $\mathcal{R}$. It is known that $\mathrm{SN}(\mathcal{R})$ and $\mathrm{QR}(\mathcal{R}, \mathscr{G})$ imply $\mathrm{SC}(\mathcal{R}, \mathscr{G})$. In our example, $\mathrm{QR}(\mathcal{R}, \mathscr{G})$ follows since all elements of $\{\mathsf{sum}(xs^{\mathrm{List}}), +(x^{\mathrm{Nat}}, y^{\mathrm{Nat}})\}\Theta$ are reducible. Thus, $\mathrm{SC}(\mathcal{R}, \mathscr{G})$. In a similar way, $\mathrm{SC}(\mathcal{R}', \mathscr{G}')$ can be checked where $\mathscr{G}' = \mathscr{G} \cup \{\mathsf{sum1}\}$.

Finally, a short remark on how inductive validity is proved automatically. In fact, this can be proved by automated inductive theorem proving methods such as *rewriting induction* [2, 13]. In our case, since $\mathrm{SN}(\mathcal{R})$, one easily verifies automatically that any equations in

$$\mathcal{E} \quad \left\{ \begin{array}{l} +(x^{\mathrm{Nat}}, 0) \quad \approx \quad +(0, x^{\mathrm{Nat}}) \\ +(+(x^{\mathrm{Nat}}, y^{\mathrm{Nat}}), z^{\mathrm{Nat}}) \\ \qquad \approx \quad +(x^{\mathrm{Nat}}, +(y^{\mathrm{Nat}}, z^{\mathrm{Nat}})) \end{array} \right.$$

are inductively valid in $\mathcal{R}$ using the rewriting induction.

To conclude, correctness proofs of TRS transformation based on developed templates are conducted in an automatic way. We are currently working toward an implementation of our program transformation framework.

7. CONCLUSION

We proposed a new framework of program transformation by templates based on term rewriting. We gave a sound and complete term pattern matching algorithm and showed that how our program transformation is automated using this algorithm. We showed how transformation templates for the correct program transformation is developed via step-by-step transformations of TRS patterns. Then we showed that in any program transformation by developed templates the correctness of the transformation is verified automatically.

We now compare our framework for the program transformation and those based on lambda calculus [4, 6, 8, 10, 11, 14].

There is no significant difference between the second-order matching algorithm by Huet and Lang [11] and ours. However, we preferred organizing the matching algorithm in the rewriting framework to encoding it based on the lambda calculus framework. Yokoyama et al. proposed a simpler and efficient matching algorithm for deterministic second-order pattern[19]. By incorporating their ideas to our framework, more efficient and useful algorithm may be found.

For the correctness proof of the transformation, a most significant difference between our approach and and those by Huet and Lang is that our approach is based on the operational semantics while Huet and Lang's one is on the denotational semantics. The basis of our correctness verification method is inductionless induction in which the Church-Rosser property and the sufficient completeness of rewriting systems play essential roles. Contrasted to this, Huet and Lang's approach is based on the fixpoint induction.

In our framework, hypothesis of transformation pattern must be satisfied by the original programs and therefore verification of hypothesis and matching algorithm are separately performed in different transformation phases. However, there are such cases that a help of hypothesis of programs is needed to find an appropriate second-order matcher. To incorporate such a mechanism into our framework remains as our future work.

Acknowledgments

The authors are grateful to anonymous referees for useful comments and advices. The authors also thank Professor Zhenjiang Hu, Professor Atsushi Togashi, and Tetsuo Yokoyama for valuable comments. This work was partially supported by grants from Japan Society for the Promotion of Science, Nos. 14580357 and 17700002, a grant from Ministry of Education, Culture, Sports, Science and Technology, No. 16016202, and a grant from International Information Science Foundation, No. 2005.1.2.876.

8. REFERENCES

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

[2] A. Bouhoula, E. Kounalis, and M. Rusinowitch. Automated mathematical induction. *Journal of Logic and Computation*, 5(5):631–668, 1995.

[3] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. *Tree Automata Techniques and Applications*. 1997.

[4] R. Curien, Z. Qian, and H. Shi. Efficient second-order matching. In *Proceedings of the 7th International Conference on Rewriting Techniques and Applications*, volume 1103 of *LNCS*, pages 317–331. Springer-Verlag, 1996.

[5] O. de Moor and G. Sittampalam. Generic program transformation. In *Proceedings of the 3rd International Summer School on Advanced Functional Programming*, volume 1608 of *LNCS*, pages 116–149. Springer-Verlag, 1999.

[6] O. de Moor and G. Sittampalam. Higher-order matching for program transformation. *Theoretical Computer Science*, 269:135–162, 2001.

[7] J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Automated termination proofs with AProVE. In *Proceedings of the 15th International Conference on Rewriting Techniques and Applications*, volume 3091 of *LNCS*, pages 210–220. Springer-Verlag, 2004.

[8] K. Hirata, K. Yamada, and M. Harao. Tractable and intractable second-order matching problems. *Journal of Symbolic Computation*, 37(5):611–628, 2004.

[9] N. Hirokawa and A. Middeldorp. Tsukuba termination tool. In *Proceedings of the 14th International Conference on Rewriting Techniques and Applications*, volume 2706 of *LNCS*, pages 311–320. Springer-Verlag, 2003.

[10] G. Huet. A unification algorithm for typed λ-calculus. *Theoretical Computer Science*, 1:27–57, 1975.

[11] G. Huet and B. Lang. Proving and applying program transformations expressed with second order patterns. *Acta Informatica*, 11:31–55, 1978.

[12] S. Kamin and J.-J. Lévy. Two generalizations of the recursive path ordering. Unpublished manuscript, University of Illinois, 1980.

[13] U. S. Reddy. Term rewriting induction. In *Proceedings of the 10th International Conference on Automated Deduction*, volume 449 of *LNAI*, pages 162–177, 1990.

[14] G. Sittampalam. *Higher-Order Matching for Program Transformation*. PhD thesis, Magdalen College, 2001.

[15] Terese. *Term Rewriting Systems*. Cambridge University Press, 2003.

[16] Y. Toyama. Commutativity of term rewriting systems. In *The Second France-Japan Artificial Intelligence and Computer Science Symposium*, 1987.

[17] Y. Toyama. How to prove equivalence of term rewriting systems without induction. *Theoretical Computer Science*, 90:369–390, 1991.

[18] Eelco Visser. A survey of rewriting strategies in program transformation systems. In *Workshop on Reduction Strategies in Rewriting and Programming*, volume 57 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers, 2001.

[19] T. Yokoyama, Z. Hu, and M. Takeichi. Deterministic second-order patterns. *Information Processing Letters*, 89(6):309–314, 2004.