A Term Rewriting Approach to Program Transformation by Templates

A Term Rewriting Approach

to

Program Transformation by Templates

by

Yuki Chiba

Supervisor: **Professor Yoshihito Toyama**

Tohoku University

February 12, 2008

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 thesis 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 correct templates and a simple method to construct such templates without explicit use of induction. We then show that in any program transformation by correct 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.

RAPT (Rewriting-based Automated Program Transformation system), which implements our framework is reported in this thesis. RAPT transforms input many-sorted TRSs according to specified correct templates and verifies its correctness automatically. We explain each phase within RAPT and report several experiments of program transformations obtained from RAPT.

To enhance the variety of program transformation, it is important to introduce new transformation templates. Up to our knowledge, however, few works discuss about the construction of transformation templates. We then propose a method that automatically constructs transformation templates from similar program transformations. The key idea of our method is a second-order generalization, which is an extension of Plotkin's first-order generalization (1969). We give a second-order generalization algorithm and prove the soundness of the algorithm. We then report about an implementation of the generalization procedure and an experiment on the construction of transformation templates.

Acknowledgments

I would like to express my gratitude to my supervisor Professor Yoshihito Toyama for his kind discussion, guidance and encouragement during this work. He has showed me a notion of equivalent transformation of TRSs which plays an essential role in this work. I also wish to thank Associate Professor Takahito Aoto for his kind discussion, guidance and encouragement. A part of RAPT has been implemented by him. I am very grateful to Professor Atsushi Ohori and Professor Naoki Kobayashi for their useful comments and suggestions.

Contents

Chapter 1

Introduction

1.1 Program Transformation by Templates

Automatically transforming given programs to optimize efficiency is one of the most fascinating techniques for programming languages [18, 19]. Several techniques for transforming functional programming languages have been developed [3, 12, 26]. Huet and Lang [12] 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 which 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 transformations using denotational semantics.

After Huet and Lang's pioneering work, Curien et al. [6] provided an improved matching algorithm using top-down matching method. Yokoyama et al. [27] presented sufficient conditions to have at most one solution and a deterministic algorithm to find such a solution. de Moor and Sittampalam [8] 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. In their framework, the correctness of transformations is often verified using several inductive properties of programs (e.g. associativity of addition) as hypotheses. It is known that one may need to verify different hypotheses to guarantee the correctness of each transformation. To the best of our knowledge, there exists no framework of program transformation by templates equipping automated verification of hypotheses to guarantee the correctness of transformations

1.2 Term Rewriting System

Term rewriting systems (TRSs, for short) are used for computational models of functional programming languages[1, 23]. TRSs consist sets of rewrite rules of terms. Let us consider an example of TRS. Formal definitions of TRSs appear in next chapter. The following TRS $\mathcal{R}_{add}$

represents a program which computes additions of two input natural numbers.

$$\mathcal{R}_{add} \quad \begin{cases} +(\mathsf{0}, x) & \to x \\ +(\mathsf{s}(x), y) & \to \mathsf{s}(+(x, y)) \end{cases}$$

Note that, natural numbers $0, 1, 2, \cdots$ are expressed as $\mathsf{0}, \mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{s}(\mathsf{0})), \cdots$, respectively.

The computation by TRSs is carried out by the reduction. For example, an addition of 2 and 3 is computed by rewriting a term $+(\mathsf{s}(\mathsf{s}(0)), \mathsf{s}(\mathsf{s}(\mathsf{s}(0))))$ using the TRS $\mathcal{R}_{add}$ as follows:

$$\begin{array}{lll} +(\mathsf{s}(\mathsf{s}(0)), \mathsf{s}(\mathsf{s}(\mathsf{s}(0)))) & \to_{\mathcal{R}_{add}} & \mathsf{s}(+(\mathsf{s}(0), \mathsf{s}(\mathsf{s}(\mathsf{s}(0))))) \\ & \to_{\mathcal{R}_{add}} & \mathsf{s}(\mathsf{s}(+(0, \mathsf{s}(\mathsf{s}(\mathsf{s}(0)))))) \\ & \to_{\mathcal{R}_{add}} & \mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(0))))) \end{array}$$

Since there exist several automated theorem proving methods based on term rewriting for verifying inductive properties of programs[21, 25], one may expect to construct a framework of TRS transformation by templates with automated verification of the correctness of transformations by applying such automated theorem proving techniques. However, no framework of TRS transformation using pattern matching is known.

1.3 Overview of this Thesis

We propose a framework of program transformation by templates in this thesis based on term rewriting. Applying automated theorem proving techniques of term rewriting, the correctness of transformations are verified automatically in our framework. Chapter 2 recalls basic notions about term rewriting which are used in this thesis.

In our framework, a transformation template (*template*, for short) consists of two term rewriting system patterns (*TRS patterns*, for short)—an input part and an output part of the template. A TRS is transformed according to a template, first by performing the pattern matching between the given TRS and the input part of the template, and then applying the result of pattern matching to the output part of the template (Figure 1.1). In Chapter 3, we explain the detail of our framework through motivating examples. We also introduce the notion of term homomorphisms to describe how a TRS pattern matches a concrete TRS.

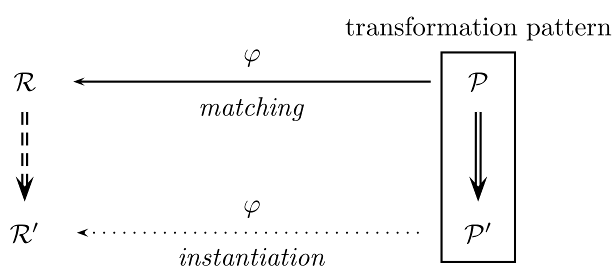


Figure 1.1: Overview of TRS transformation by templates

Chapter 4 discusses about verifying the correctness of transformations. In contrast to existing works, the correctness of transformations is discussed based on operational semantics. To guarantee the correctness of transformation, we introduce the notion of correct templates and a simple method of constructing such templates without explicit use of induction. We also give sufficient conditions to guarantee the correctness of transformations by correct templates. We then show that the correctness of transformation can be verified automatically for some class of TRSs.

A key part of our procedure of TRS transformation using templates— the TRS pattern matching problem—is solved using the term pattern matching algorithm **Match** introduced in Chapter 5. We then show termination, soundness and completeness of **Match**. We also extend **Match** to solve TRS pattern matching problems.

In Chapter 6, we explain about RAPT (Rewriting-based Automated Program Transformation system), which implements our framework and reports several experiments using RAPT.

In order to apply our framework of program transformation, one have to construct transformation templates beforehand. Since transformation templates are often constructed by generalizing similar transformations, a generalization procedure can help to construct transformation templates automatically. Therefore, we propose 2nd-order generalization algorithm to construct transformation templates automatically in Chapter 7.

Chapter 8 concludes this thesis and reports differences against existing works.

Chapter 2

Term Rewriting System

This chapter introduces basic notions of term rewriting systems used in this thesis based on [1].

Let $\mathscr{F}$ and $\mathscr{V}$ be sets 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 $\mathrm{T}(\mathscr{F}, \mathscr{V})$ of *terms* inductively by:

1. $\mathscr{V} \subseteq \mathrm{T}(\mathscr{F}, \mathscr{V})$; and
2. $f(t_1, \ldots, t_n) \in \mathrm{T}(\mathscr{F}, \mathscr{V})$ for any $f \in \mathscr{F}$ such that $\mathrm{arity}(f) = n$ and $t_1, \ldots, t_n \in \mathrm{T}(\mathscr{F}, \mathscr{V})$.

A term without variables is a *ground* term. The set of ground terms is denoted by $\mathrm{T}(\mathscr{F})$. 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 $\mathrm{T}(\mathscr{F}, \mathscr{V})$. A substitution θ is extended to a mapping $\hat{\theta}$ over terms $\mathrm{T}(\mathscr{F}, \mathscr{V})$ 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 $\mathrm{T}(\mathscr{F} \cup \{\Box_i \mid i \geq 1\}, \mathscr{V})$. $C[s_1, \ldots, s_n]$ 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 term obtained by replacing each $\Box_i$ in C with s_i (*indexed replacement*). A context C with precisely one hole is denoted by $C[\,]$. The set of contexts is denoted by $\mathrm{T}^{\Box}(\mathscr{F}, \mathscr{V})$; its subset $\mathrm{T}(\mathscr{F} \cup \{\Box_i \mid 1 \leq i \leq n\}, \mathscr{V})$ is denoted by $\mathrm{T}^{\Box}_n(\mathscr{F}, \mathscr{V})$. $\mathrm{T}^{\Box}(\mathscr{F})$ and $\mathrm{T}^{\Box}_n(\mathscr{F})$ are defined in the same way as $\mathrm{T}(\mathscr{F})$.

Example 2.1 (Context)**.** Let $C_1 = \mathsf{f}(\Box_1)$, $C_2 = \mathsf{g}(\Box_2, \Box_1)$, and $C_3 = \mathsf{g}(\Box_2, \mathsf{g}(\Box_1, \Box_1))$ be contexts. Here, we get

$$\begin{aligned}
C_1[\mathsf{a}] &= \mathsf{f}(\mathsf{a}) \\
C_1\langle \mathsf{a}, \mathsf{b}\rangle &= \mathsf{f}(\mathsf{a}) \\
C_2[\mathsf{a}, \mathsf{b}] &= \mathsf{g}(\mathsf{a}, \mathsf{b}) \\
C_2\langle \mathsf{a}, \mathsf{b}\rangle &= \mathsf{g}(\mathsf{b}, \mathsf{a}) \\
C_3[\mathsf{a}, \mathsf{b}, \mathsf{c}] &= \mathsf{g}(\mathsf{a}, \mathsf{g}(\mathsf{b}, \mathsf{c})) \\
C_3\langle \mathsf{a}, \mathsf{b}\rangle &= \mathsf{g}(\mathsf{b}, \mathsf{g}(\mathsf{a}, \mathsf{a}))
\end{aligned}$$

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 $t = C[r\theta]$.

Example 2.2 (Summation)**.** The following TRS $\mathcal{R}_{sum}$ represents a program which computes summations of input lists of natural numbers.

$$\mathcal{R}_{sum}\quad \begin{cases} \mathsf{sum}([\;]) & \to & \mathsf{0} \\ \mathsf{sum}(x_1{:}y_1) & \to & +(x_1, \mathsf{sum}(y_1)) \\ +(\mathsf{0}, x_2) & \to & x_2 \\ +(\mathsf{s}(x_3), y_3) & \to & \mathsf{s}(+(x_3, y_3)) \end{cases}$$

Note that, natural numbers $0, 1, 2, \cdots$ are expressed as $\mathsf{0}, \mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{s}(\mathsf{0})), \cdots$, respectively.

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* of a TRS $\mathcal{R}$ when $s \to_{\mathcal{R}} t$ for no term t. NF($\mathcal{R}$) denotes the set of terms in normal form of a TRS $\mathcal{R}$.

A TRS $\mathcal{R}$ is *terminating*, or *strongly normalizing* (SN($\mathcal{R}$)) if there exists no infinite reduction $s_1 \to_{\mathcal{R}} s_2 \to_{\mathcal{R}} s_3 \to_{\mathcal{R}} \cdots$. A binary relation $>$ is *well-founded* if there exists no infinite sequence such that $a_1 > a_2 > \cdots$. A binary relation $>$ of terms is *closed under contexts* if $s > t$ implies $C[s] > C[t]$, for any terms s and t and contexts $C[\;]$. A binary relation $>$ of terms is *closed under substitutions* if $s > t$ implies $s\theta > t\theta$, for any terms s and t and substitutions θ. A *strict order* is a transitive and irreflexive relation. A *reduction order* is a well-founded strict order which is closed under contexts and substitutions. The following theorem shows the motivation for introducing reduction orders:

Theorem 2.3. *A TRS $\mathcal{R}$ is terminating iff there exists a reduction order $>$ such that $l > r$ for all $l \to r \in \mathcal{R}$.*

Proof.

1. Assume $\mathcal{R}$ is terminating. It is obvious that $\xrightarrow{+}_{\mathcal{R}}$ is a reduction order and $l \xrightarrow{+}_{\mathcal{R}} r$ for all $l \to r \in \mathcal{R}$.

2. Since $>$ is a reduction order, for any $l \to r \in \mathcal{R}$, $l > r$ implies $C[l\theta] > C[r\theta]$, for any contexts $C[\;]$ and substitutions θ. Thus, $s_1 \to_{\mathcal{R}} s_2$ implies $s_1 > s_2$. Since $>$ is well-founded, there exists no infinite reduction $s_1 \to_{\mathcal{R}} s_2 \to_{\mathcal{R}} s_3 \to_{\mathcal{R}} \cdots$.

$\square$

Definition 2.4 (lexicographic path order)**.** *Let $\mathscr{F}$ be a finite set of function symbols and $>$ be a strict order on $\mathscr{F}$. The* lexicographic path order *$>_{lpo}$ on $\mathrm{T}(\mathscr{F}, \mathscr{V})$ induced by $>$ is defined as follows:*
$s >_{lpo} t$ iff

1. *$t \in \mathscr{V}(s)$ and $s \neq t$, or*
2. *$s = f(s_1, \cdots, s_m)$, $t = g(t_1, \cdots, t_n)$, and*
   (a) *there exists i, $1 \leq i \leq m$, with $s_i >_{lpo} t$, or*
   (b) *$f > g$ and $s >_{lpo} t_j$ for all j, $1 \leq j \leq n$, or*

(c) $f = g$, $s >_{lpo} t_j$ for all j, $1 \leq j \leq n$, and there exists i, $1 \leq i \leq m$, such that $s_1 = t_1, \cdots, s_{i-1} = t_{i-1}$ and $s_i >_{lpo} t_i$.

Theorem 2.5. *For any strict order on $\mathscr{F}$, the induced lexicographic path order $>_{lpo}$ is a reduction order on* $\mathrm{T}(\mathscr{F}, \mathscr{V})$.

A TRS $\mathcal{R}$ is *confluent*, or has the *Church-Rosser property*, (denoted by $\mathrm{CR}(\mathcal{R})$) 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 *locally confluent*, or has the *weakly Church-Rosser property*, (denoted by $\mathrm{WCR}(\mathcal{R})$) if, for any term s, s_1, s_2, $s \to_{\mathcal{R}} s_1$ and $s \to_{\mathcal{R}} s_2$ imply that there exists a term t such that $s_1 \xrightarrow{*}_{\mathcal{R}} t$ and $s_2 \xrightarrow{*}_{\mathcal{R}} t$. The following lemma is a variant of Newman's Lemma [16].

Lemma 2.6. *A terminating TRS is confluent if it is locally confluent.*

For substitutions σ and θ, we say σ is *more general* than θ if there exists a substitution σ' such that $\theta = \sigma' \circ \sigma$. In this case, we write $\sigma \lesssim \theta$. Terms s and t are *unifiable* if there exists substitution σ such that $s\sigma = t\sigma$. In this case σ is a *unifier* of s and t. A unifier σ of terms s and t is *most general unifier* if σ is more general than any unifier of s and t. A most general unifier of s and t is denoted as $\mathrm{mgu}(s, t)$.

Definition 2.7. *Let $l_1 \to r_1$ and $l_2 \to r_2$ be rewrite rules whose variables are disjoint (i.e. $\mathscr{V}(l_1) \cap \mathscr{V}(l_2) = \emptyset$). If there exists a context $C[\ ]$ such that $l_1 = C[l_1']$ where l_1' is not a variable and l_1' and l_2 are unifiable, we say l_1* overlaps *l_2 and a pair of terms $\langle r_1\sigma, C[r_2]\sigma\rangle$ is called a* critical pair *where σ is a most general unifier of l_1' and l_2.*

The critical pairs of a TRS $\mathcal{R}$ are the critical pairs between any two rules whose variables are renamed. The set of critical pairs of a TRS $\mathcal{R}$ is denoted by $\mathrm{CP}(\mathcal{R})$. We say that a TRS $\mathcal{R}_1$ overlaps a TRS $\mathcal{R}_2$ if there exist rewrite rules $l_1 \to r_1 \in \mathcal{R}_1$ and $l_2 \to r_2 \in \mathcal{R}_2$ such that l_1 overlaps l_2. A critical pair $\langle s, t\rangle$ of a TRS $\mathcal{R}$ is *joinable* if there exists a term u such that $s \xrightarrow{*}_{\mathcal{R}} u$ and $t \xrightarrow{*}_{\mathcal{R}} u$.

The following is called *Critical Pair Theorem*, which is brought by Knuth and Bendix[14].

Theorem 2.8. *A TRS is locally confluent iff all its critical pairs are joinable.*

We obtain the following theorem from Critical Pair Theorem and Lemma 2.6.

Corollary 2.9. *A terminating TRS is confluent iff all its critical pairs are joinable*

Critical Pair Theorem can apply only terminating TRSs to show their confluence and confluence is undecidable in general. It is known that there are sufficient conditions to guarantee confluence of TRSs. Orthogonality is one of such sufficient conditions. 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. 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. A TRS is *orthogonal* if it is left-linear and has no critical pairs. The following theorem shows that orthogonality is a sufficient condition to show confluence.

Theorem 2.10. *If a TRS is orthogonal then it is confluent.*

We note that Theorem 2.10 can apply nonterminating TRSs.

Modularity is one of effective methods to show several properties. Toyama showed that confluence has modularity[24, 25].

Theorem 2.11. *Let $\mathcal{R}_1$ and $\mathcal{R}_2$ be left-linear TRSs such that each TRS does not overlap another.* $\mathrm{CR}(\mathcal{R}_1) \wedge \mathrm{CR}(\mathcal{R}_2)$ *implies* $\mathrm{CR}(\mathcal{R}_1 \cup \mathcal{R}_2)$.

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

Definition 2.12. *Suppose $\mathscr{F}_{\mathrm{c}} \subseteq \mathscr{G} \subseteq \mathscr{F}$. A TRS $\mathcal{R}$ is* sufficiently complete for $\mathscr{G}$ *(*$\mathrm{SC}(\mathcal{R}, \mathscr{G})$*) when for any ground term $s \in \mathrm{T}(\mathscr{G})$ there exists $t \in \mathrm{T}(\mathscr{F}_{\mathrm{c}})$ such that $s \xrightarrow{*}_{\mathcal{R}} t$.*

A TRS $\mathscr{R}$ is *quasi-reducible* if for any $f \in \mathscr{D}$ ($\mathrm{arity}(f) = n$) and $s_1, \ldots, s_n \in T(\mathscr{C})$ $f(s_1, \ldots, s_n)$ is not a normal form of $\mathcal{R}$. Note that if a TRS $\mathcal{R}$ is sufficient complete then $\mathcal{R}$ is quasi-reducible.

Proposition 2.13. $\mathrm{SN}(\mathcal{R})$ *and* $\mathrm{QR}(\mathcal{R}, \mathscr{G})$ *imply* $\mathrm{SC}(\mathcal{R}, \mathscr{G})$.

Let $\mathcal{S}$ be a set of *sorts*. $\mathscr{V}^{\beta}$ denote a set of variables whose sorts are β. $\mathscr{F}^{\alpha_1 \times \cdots \times \alpha_n \to \beta}$ denotes the set of function symbols which take arguments of sorts $\alpha_1, \cdots, \alpha_n$ to values of sorts β. We note that for any function symbol f, $\mathrm{arity}(f) = n$ implies $f \in \mathscr{F}^{\alpha_1 \times \cdots \times \alpha_n \to \beta}$ for some $\alpha_1, \cdots, \alpha_n, \beta \in \mathcal{S}$. We now define the set $\mathrm{T}^{\beta}(\mathscr{F}, \mathscr{V})$ of *many-sorted terms* whose sorts are β inductively by:

1. $\mathscr{V}^{\beta} \subseteq \mathrm{T}^{\beta}(\mathscr{F}, \mathscr{V})$
2. $f(t_1, \cdots, t_n) \in \mathrm{T}^{\beta}(\mathscr{F}, \mathscr{V})$ for any $f \in \mathscr{F}^{\alpha_1 \times \cdots \times \alpha_n \to \beta}$ and $t_i \in \mathrm{T}^{\alpha_i}(\mathscr{F}, \mathscr{V})$ for all i $(1 \leq i \leq n)$.

A TRS $\mathcal{R}$ is called a *many-sorted TRS* if for any $l \to r \in \mathcal{R}$, $l \in \mathrm{T}^{\beta}(\mathscr{F}, \mathscr{V})$ iff $r \in \mathrm{T}^{\beta}(\mathscr{F}, \mathscr{V})$ for some $\beta \in \mathcal{S}$. We define $\mathrm{SN}(\mathcal{R})$, $\mathrm{CR}(\mathcal{R})$, $\mathrm{SC}(\mathcal{R}, \mathscr{G})$ and $\mathcal{QR}(\mathrm{R}, \mathscr{G})$

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 substitution θ is *ground on* $\mathscr{G}$ if $\theta(x) \in \mathrm{T}(\mathscr{G})$ for any $x \in \mathrm{dom}(\theta)$. An equation $s \approx t$ is an *inductive consequence of* $\mathcal{R}$ *for* $\mathscr{G}$ ($\mathcal{R}, \mathscr{G} \vdash_{ind} s \approx t$) when for any ground substitution θ_g on $\mathscr{G}$ such that $\mathscr{V}(s) \cup \mathscr{V}(t) \subseteq \mathrm{dom}(\theta_g)$, $s\theta_g \xleftrightarrow{*}_{\mathcal{R}} t\theta_g$ 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 equivalence of two TRSs are defined as follows:

Definition 2.14. *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 $s \in \mathrm{T}(\mathscr{G})$ and ground constructor term $t \in \mathrm{T}(\mathscr{F}_{\mathrm{c}})$, $s \xrightarrow{*}_{\mathcal{R}} t$ 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 $s \xleftrightarrow{*}_{\mathcal{R}} t$ iff $s \xleftrightarrow{*}_{\mathcal{R}'} t$ for all ground terms $s \in \mathrm{T}(\mathscr{F})$ and ground constructor term $t \in \mathrm{T}(\mathscr{F}_{\mathrm{c}})$. 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.

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 [25] for some restricted class of TRSs. We simplify and improve this technique for our framework in Chapter 4.

Chapter 3

Program Transformation by Templates

In this chapter, we formalize the framework of program transformation by templates based on term rewriting. We give a notion of transformation templates within our framework. We then introduce a notion of *term homomorphism* to specify how to apply transformation templates to TRSs. We also show that term homomorphisms preserve reductions.

3.1 Motivating Example

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

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

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 $\mathcal{R}_{cat}$.

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

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.

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

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

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

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

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

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 $\mathcal{R}_{sum}$ and $\mathcal{R}'_{sum}$ ($\mathcal{R}_{cat}$ and $\mathcal{R}'_{cat}$) is guaranteed when the instantiation satisfies the following equations, called a *hypothesis*:

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

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* arity(p)*). A* term pattern *(or just* pattern*) is a term in* $\mathrm{T}(\mathscr{F} \cup \mathscr{X}, \mathscr{V})$*. 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.*

3.2 Term Homomorphism

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 [5]—we call this a *term homomorphism*.

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

1. *$\varphi(p) \in T^{\Box}_{\mathrm{arity}(p)}(\mathscr{F})$ 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 $\mathrm{T}(\mathscr{F} \cup \mathscr{X}, \mathscr{V})$ as follows:

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

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,

$$\begin{array}{rcl}\varphi(t\theta) & = & \varphi(f(t_1\theta,\ldots,t_n\theta)) \\ & = & f(\varphi(t_1\theta),\ldots,\varphi(t_n\theta)) \\ & = & f(\varphi(t_1)\varphi(\theta),\ldots,\varphi(t_n)\varphi(\theta))\end{array}$$

$$\begin{array}{rcl} & = & f(\varphi(t_1),\ldots,\varphi(t_n))\varphi(\theta) \\ & = & \varphi(f(t_1,\ldots,t_n))\varphi(\theta) \\ & = & \varphi(t)\varphi(\theta).\end{array}$$

3. $t = p(t_1,\ldots,t_n)$ with $p \in \mathscr{X}$.

   Then,

$$\begin{array}{rl} & \varphi(t\theta) \\ = & \varphi(p(t_1,\ldots,t_n)\theta) \\ = & \varphi(p(t_1\theta,\ldots,t_n\theta)) \\ = & \varphi(p)\langle\varphi(t_1\theta),\ldots,\varphi(t_n\theta)\rangle \\ = & \varphi(p)\langle\varphi(t_1)\varphi(\theta),\ldots,\varphi(t_n)\varphi(\theta)\rangle \\ = & (\varphi(p)\langle\varphi(t_1),\ldots,\varphi(t_n)\rangle)\varphi(\theta) \\ = & (\varphi(p(t_1,\ldots,t_n))\varphi(\theta) \\ = & \varphi(t)\varphi(\theta).\end{array}$$

   (Note that $\mathscr{V}(\varphi(p)) = \emptyset$.) $\square$

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,

$$\begin{array}{rl} & \varphi(f(s_1,\ldots,C'[t],\ldots,s_n)) \\ = & f(\varphi(s_1),\ldots,\varphi(C'[t]),\ldots,\varphi(s_n))) \\ = & f(\varphi(s_1),\ldots,\varphi(C')[\varphi(t),\ldots,\varphi(t)], \\ & \qquad\ldots,\varphi(s_n))) \\ = & f(\varphi(s_1),\ldots,\varphi(C'),\ldots,\varphi(s_n)) \\ & \qquad[\varphi(t),\ldots,\varphi(t)] \\ = & \varphi(C)[\varphi(t),\ldots,\varphi(t)]\end{array}$$

3. $C[\,] = p(s_1,\ldots,s_n)$ with $s_i = C'[\,]$ and $p \in \mathscr{X}$.

   Then,

$$\begin{array}{rl} & \varphi(p(s_1,\ldots,C'[t],\ldots,s_n)) \\ = & \varphi(p)\langle\varphi(s_1),\ldots,\varphi(C'[t]),\ldots,\varphi(s_n)\rangle \\ = & \varphi(p)\langle\varphi(s_1),\ldots, \\ & \quad\varphi(C')[\varphi(t),\ldots,\varphi(t)],\ldots,\varphi(s_n)\rangle \\ = & (\varphi(p)\langle\varphi(s_1),\ldots,\varphi(C'),\ldots,\varphi(s_n)\rangle) \\ & \qquad[\varphi(t),\ldots,\varphi(t)] \\ = & \varphi(p(s_1,\ldots,C',\ldots,s_n)) \\ & \qquad[\varphi(t),\ldots,\varphi(t)] \\ = & \varphi(C)[\varphi(t),\ldots,\varphi(t)].\end{array}$$

$\square$

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

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

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

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 $\mathcal{R}'_{sum} = \varphi(\mathcal{P}')$.

$$\varphi = \left\{ \begin{array}{ll}
\mathsf{f} \mapsto \mathsf{sum}(\square_1), & u_1 \mapsto x_1, u_6 \mapsto x_6, \\
\mathsf{g} \mapsto +(\square_1, \square_2), & v_1 \mapsto y_1, v_6 \mapsto y_6, \\
\mathsf{f}_1 \mapsto \mathsf{sum1}(\square_1, \square_2), & u_2 \mapsto x_2, w_6 \mapsto z_6, \\
\mathsf{a} \mapsto [\,], & v_3 \mapsto x_3, u_7 \mapsto x_7, \\
\mathsf{b} \mapsto \mathsf{0}, & w_3 \mapsto y_3, v_8 \mapsto y_8, \\
\mathsf{c} \mapsto \square_1 : \square_2, & u_4 \mapsto x_4, w_8 \mapsto z_8 \\
\mathsf{d} \mapsto \mathsf{s}(\square_2), & u_5 \mapsto x_5,
\end{array} \right\}$$

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 chapter, we will discuss the criteria for the templates for the correct transformation and try to give a definite answer to this question.

3.3 Summary

We proposed a framework of program transformation by templates in this chapter. Programs are represented by TRSs. A transformation templates is defined as a triple $\langle \mathcal{P}, \mathcal{P}', \mathcal{H} \rangle$ of two TRS patterns $\mathcal{P}$ and $\mathcal{P}'$ which are TRSs including pattern variables and hypothesis $\mathcal{H}$. In order to specify how a template is applied to a concrete TRS, we introduced a notion of term homomorphisms. We then show that term homomorphisms preserve reductions (Proposition 3.8). The definition of TRS transformations by templates was given in Definition 3.9.

Chapter 4

Correctness of Transformations

Verifying the correctness is one of important problems for program transformations. The correctness of transformations is formalized as the equality of input and output TRSs in our framework. Equivalent transformation of TRSs proposed by Toyama[25] is one of techniques to verify the equality of two TRSs. In this chapter, we simplify and improve this technique to specialize in our framework. We then propose a notion of correct templates which guarantee the correctness of transformation of restricted TRSs. The method of constructing correct templates is given by lifting up the notion of equivalent transformation of TRSs to the template level.

4.1 Equivalent Transformation of TRS

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.

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 [25] for some restricted class of TRSs. Let us 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 $\mathrm{T}(\mathscr{G}, \mathscr{V})$.

Definition 4.1. *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 $r \in \mathrm{T}(\mathscr{F}_k, \mathscr{V})$. 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 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 $\mathcal{R} \underset{I}{\overset{*}{\Rightarrow}} \cdot \underset{A}{\overset{*}{\Rightarrow}} \cdot \underset{E}{\overset{*}{\Rightarrow}} \mathcal{R}'$ under $\mathcal{E}$.*

Differences against [25] are listed as follows:

1. Orders of applying inference rules are fixed (*Introduction* $\to$ *Addition* $\to$ *Elimination*).
2. Some equations can be used in the *Addition* rule.

Theorem 4.2. *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 $\mathcal{R}, \mathscr{G} \vdash_{ind} \mathcal{E}$ 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 the 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 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. 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 $\mathrm{T}(\mathscr{G} \cap \mathscr{G}')$.

1. $\overset{*}{\leftrightarrow}_{\mathcal{R}} = \overset{*}{\leftrightarrow}_{\mathcal{R}_I}$ on $\mathrm{T}(\mathscr{G})$. (i.e., for any $s, t \in \mathrm{T}(\mathscr{G})$, $s \overset{*}{\leftrightarrow}_{\mathcal{R}} t$ iff $s \overset{*}{\leftrightarrow}_{\mathcal{R}_I} t$.)
   ($\subseteq$) Trivial. ($\supseteq$) Suppose that $s \overset{*}{\leftrightarrow}_{\mathcal{R}_I} t$ where $s, t \in \mathrm{T}(\mathscr{G})$. By $\mathrm{SC}(\mathcal{R},\mathscr{G})$, there exist ground constructor terms $s', t' \in \mathrm{T}(\mathscr{F}_{\mathrm{c}})$ 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 $\mathrm{T}(\mathscr{F}_{\mathrm{c}}) \subseteq \mathrm{NF}(\mathcal{R}_I)$, $s' = t'$ 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 $\mathrm{T}(\mathscr{F})$. (i.e., for any $s, t \in \mathrm{T}(\mathscr{F})$, $s \overset{*}{\leftrightarrow}_{\mathcal{R}_I} t$ iff $s \overset{*}{\leftrightarrow}_{\mathcal{R}_A} t$.)
   ($\subseteq$) Trivial. ($\supseteq$) Suppose that $s \leftrightarrow_{\mathcal{E}} t$ where $s, t \in \mathrm{T}(\mathscr{F})$. 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 $\theta_g(x) \overset{*}{\to}_{\mathcal{R}_I} \theta_g^{\mathrm{c}}(x) \in \mathrm{T}(\mathscr{F}_{\mathrm{c}})$ for any $x \in \mathrm{dom}(\theta_g)$. Then, $C[l\theta_g] \overset{*}{\to}_{\mathcal{R}_I} C[l\theta_g^{\mathrm{c}}]$ and $C[r\theta_g] \overset{*}{\to}_{\mathcal{R}_I} C[r\theta_g^{\mathrm{c}}]$ hold. Now, since $l\theta_g^{\mathrm{c}}, r\theta_g^{\mathrm{c}} \in \mathrm{T}(\mathscr{G})$, we have $l\theta_g^{\mathrm{c}} \overset{*}{\leftrightarrow}_{\mathcal{R}} r\theta_g^{\mathrm{c}}$ by our assumption $\mathcal{R}, \mathscr{G} \vdash_{ind} \mathcal{E}$. Thus, by $\mathcal{R} \subseteq \mathcal{R}_I$, $C[l\theta_g^{\mathrm{c}}] \overset{*}{\leftrightarrow}_{\mathcal{R}_I} C[r\theta_g^{\mathrm{c}}]$ holds. Hence, $\overset{*}{\leftrightarrow}_{\mathcal{R}_I} \supseteq \leftrightarrow_{\mathcal{E}}$ on $\mathscr{F}$. 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 $\mathrm{T}(\mathscr{F}, \mathscr{V})$. Hence, $\overset{*}{\leftrightarrow}_{\mathcal{R}_A} \subseteq \overset{*}{\leftrightarrow}_{\mathcal{E} \cup \mathcal{R}_I} \subseteq \overset{*}{\leftrightarrow}_{\mathcal{R}_I}$ on $\mathrm{T}(\mathscr{F})$.

3. $\overset{*}{\leftrightarrow}_{\mathcal{R}_I} = \overset{*}{\leftrightarrow}_{\mathcal{R}'}$ on $\mathrm{T}(\mathscr{G}')$ (i.e., for any $s, t \in \mathrm{T}(\mathscr{G}')$, $s \overset{*}{\leftrightarrow}_{\mathcal{R}_I} t$ iff $s \overset{*}{\leftrightarrow}_{\mathcal{R}'} t$.)
   ($\supseteq$) It easily follows from item 2 and the definition of the Elimination rule. ($\subseteq$) Suppose

that $s \xleftrightarrow{*}_{\mathcal{R}_I} t$ where $s, t \in \mathrm{T}(\mathscr{G}')$. From $\mathrm{SC}(\mathcal{R}', \mathscr{G}')$, there exist ground constructor terms $s', t' \in \mathrm{T}(\mathscr{F}_c)$ such that $s \xrightarrow{*}_{\mathcal{R}'} s'$ and $t \xrightarrow{*}_{\mathcal{R}'} t'$. As we have already shown $\xleftrightarrow{*}_{\mathcal{R}_I} \supseteq \xleftrightarrow{*}_{\mathcal{R}'}$ on $\mathrm{T}(\mathscr{G}')$, it follows that $s' \xleftrightarrow{*}_{\mathcal{R}_I} s \xleftrightarrow{*}_{\mathcal{R}_I} t \xleftrightarrow{*}_{\mathcal{R}_I} t'$. From $\mathrm{CR}(\mathcal{R}_I)$ and $\mathrm{T}(\mathscr{F}_c) \subseteq \mathrm{NF}(\mathcal{R}_I)$, $s' = t'$ holds. This means $s \xrightarrow{*}_{\mathcal{R}'} s' = t' \xleftarrow{*}_{\mathcal{R}'} t$.

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

Example 4.3. Let $\mathcal{R}_{sum}$, $\mathcal{R}'_{sum}$ be the TRSs in Example 3.1. Let $\mathcal{E}$ be the following set of equations.

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

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 4.2, it follows that $\mathcal{R}_{sum} \simeq_{\mathscr{G}} \mathcal{R}'_{sum}$.

4.2 Correctness of Templates

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, $\varphi(\mathcal{P}) \underset{I}{\not\Rightarrow} \varphi(\mathcal{P}')$ when $\varphi = \{\mathsf{p} \mapsto \mathsf{f}(\square_1), \mathsf{q} \mapsto \mathsf{f}(\square_1), \mathsf{a} \mapsto \mathsf{0}, \mathsf{b} \mapsto \mathsf{1}\}$. The key idea in the proof of Theorem 4.2 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.

Correct templates are constructed by inference rules similar to equivalent transformations.

Definition 4.4. *Let $\mathcal{P}_0$ be a TRS pattern over a set $\Sigma_0 \subseteq \mathscr{F} \cup \mathscr{X}$ and $\mathcal{H}$ a hypothesis over Σ_0. A correct transformation sequence under $\mathcal{H}$ is a sequence $\mathcal{P}_0, \ldots, \mathcal{P}_n$ of TRS patterns (over $\Sigma_0, \ldots, \Sigma_n$, respectively) such that $\mathcal{P}_{k+1}$ is obtained from $\mathcal{P}_k$ by applying one of the following inference rules:*

(I) Introduction

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

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

(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

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

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

Since, for any term homomorphism φ, $\mathcal{P} \Rightarrow \mathcal{P}'$ does not imply $\varphi(\mathcal{P}) \Rightarrow \varphi(\mathcal{P}')$, generally, some restrictions of term homomorphisms are necessary to use for correct transformations. Such restrictions are needed to guarantee the fact that the Introduction rule preserves the Church-Rosser property and sufficient completeness. Let $\mathcal{P}$ and $\mathcal{P}'$ be TRS patterns over Σ and Σ', respectively and φ a term homomorphism. Suppose $\mathcal{P} \underset{I}{\Rightarrow} \mathcal{P} \cup \{p(x_1, \ldots, x_n) \to r\} = \mathcal{P}'$ by the Introduction rule. $\mathrm{CR}(\mathcal{P})$ implies $\mathrm{CR}(\mathcal{P}')$ because p does not appear in $\mathcal{P}$. If $p \notin \mathrm{dom}_{\mathscr{X}}(\varphi)$ and p does not appear in $\varphi(\mathcal{P})$, then $\mathrm{CR}(\varphi(\mathcal{P}))$ implies $\mathrm{CR}(\varphi(\mathcal{P}'))$. Further, $\mathrm{SC}(\mathcal{P}, \Sigma)$ implies $\mathrm{SC}(\mathcal{P}', \Sigma \cup \{p\})$ because any ground term pattern which contains p can be reduced to a ground term pattern which does not contain p. If $\varphi(r) \in \mathrm{T}(\mathscr{G}, \mathscr{V})$ whenever $\varphi(\mathcal{P})$ is a TRS over $\mathscr{G} \subseteq \mathscr{F}$, then $\mathrm{SC}(\varphi(\mathcal{P}), \mathscr{G})$ implies $\mathrm{SC}(\varphi(\mathcal{P}'), \mathscr{G} \cup \{p\})$. These conditions are summarized by the following definition.

Definition 4.5. *Let Σ and $\mathscr{G}$ be sets such that $\Sigma \subseteq \mathscr{F} \cup \mathscr{X}$ and $\mathscr{G} \subseteq \mathscr{F}$. A term homomorphism φ* carries Σ to $\mathscr{G}$ *if*

1. $\mathrm{dom}_{\mathscr{X}}(\varphi) = \Sigma \setminus \mathscr{F}$, *and*
2. $\mathrm{range}_{\mathscr{X}}(\varphi) \subseteq \mathscr{G}$.

where $\mathrm{range}_{\mathscr{X}}(\varphi) = \bigcup_{p \in \mathrm{dom}_{\mathscr{X}}(\varphi)} \mathscr{F}(\varphi(p))$.

Next lemma can be shown in a straightforward way.

Lemma 4.6. *Let $\mathcal{P}$ and $\mathcal{P}'$ be TRS patterns over Σ and Σ', respectively, and φ a term homomorphism which carries Σ to $\mathscr{G} \subseteq \mathscr{F}$. If $\varphi(\mathcal{P})$ is a TRS over $\mathscr{G}$ and $\mathcal{P} \underset{I}{\Rightarrow} \mathcal{P}'$ by the Introduction rule, then $\varphi(\mathcal{P}) \underset{I}{\Rightarrow} \varphi(\mathcal{P}')$ by the Introduction rule.*

In Lemma 4.6, each pattern variable appearing in $\varphi(\mathcal{P}')$ is regarded as a fresh function symbol. This is also applied to the next theorem.

We then obtain the following theorem from Proposition 3.8 and Lemma 4.6.

Theorem 4.7. *Let $\langle \mathcal{P}, \mathcal{P}', \mathcal{H} \rangle$ be a correct template where $\mathcal{P}$ and $\mathcal{P}'$ are TRS patterns over Σ and Σ', respectively, and φ a term homomorphism which carries Σ to $\mathscr{G}$. If $(\Sigma' \setminus \Sigma) \cap \mathscr{G} = \emptyset$ and $\varphi(\mathcal{P})$ is a TRS over $\mathscr{G}$, then there exists an equivalent transformation $\varphi(\mathcal{P}) \overset{*}{\Rightarrow} \varphi(\mathcal{P}')$ under $\varphi(\mathcal{H})$.*

This theorem leads to the next definition of TRS transformation by correct templates.

Definition 4.8. *Let $\langle \mathcal{P}, \mathcal{P}', \mathcal{H} \rangle$ be a template where $\mathcal{P}$ and $\mathcal{P}'$ are TRS patterns over Σ and Σ', respectively. A TRS $\mathcal{R}$ over $\mathscr{G}$ is transformed to a TRS $\mathcal{R}'$ over $\mathscr{G}'$ if there exist a term homomorphism φ and a TRS $\mathcal{R}_{com}$ over $\mathscr{G}$ such that:*

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

Next theorem gives sufficient conditions to guarantee the correctness of TRS transformations by correct templates.

Theorem 4.9. *If a left-linear CS $\mathcal{R}$ over $\mathscr{G}$ is transformed to a TRS $\mathcal{R}'$ over $\mathscr{G}'$ by a correct template $\langle \mathcal{P}, \mathcal{P}', \mathcal{H} \rangle$, then* $\mathrm{CR}(\mathcal{R}) \wedge \mathrm{SC}(\mathcal{R}, \mathscr{G}) \wedge \mathrm{SC}(\mathcal{R}', \mathscr{G})$ *implies* $\mathcal{R} \simeq_{\mathscr{G} \cap \mathscr{G}'} \mathcal{R}'$.

In this section, we give correct transformation templates which can be used to apply tupling transformations.

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

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

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

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

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

2. Let $\mathcal{P}_1 = \mathcal{P}_0 \cup \{\mathsf{r}(x) \to \langle \mathsf{p}(\mathsf{b}(x)), \mathsf{p}(x) \rangle\}$. Here, r is a fresh pattern variable. Thus, $\mathcal{P}_0 \Rightarrow \mathcal{P}_1$ by the Introduction rule.

3. Let $\mathcal{P}_2 = \mathcal{P}_1 \cup \{\mathsf{r1}(x) \to \langle \mathsf{q}(\pi_1(x), \pi_2(x)), \pi_1(x) \rangle\}$. Here, $\mathsf{r1}$ is a fresh pattern variable. Thus, $\mathcal{P}_1 \Rightarrow \mathcal{P}_2$ by the Introduction rule.

4. Let $\mathcal{P}_3 = \mathcal{P}_2 \cup \{\mathsf{r}(\mathsf{a}) \to \langle \mathsf{d}, \mathsf{c} \rangle\}$. Here, we have $\mathsf{r}(a) \to_{\mathcal{P}_2} \langle \mathsf{p}(\mathsf{b}(\mathsf{a})), \mathsf{p}(\mathsf{a}) \rangle \xrightarrow{*}_{\mathcal{P}_2} \langle \mathsf{d}, \mathsf{c} \rangle$. Thus, $\mathcal{P}_2 \Rightarrow \mathcal{P}_3$ by the Addition rule.

5. Let $\mathcal{P}_4 = \mathcal{P}_3 \cup \{\mathsf{r}(\mathsf{b}(x)) \to \mathsf{r1}(\mathsf{r}(x))\}$. Here, we have $\mathsf{r}(\mathsf{b}(x)) \to_{\mathcal{P}_3} \langle \mathsf{p}(\mathsf{b}(\mathsf{b}(x))), \mathsf{p}(\mathsf{b}(x)) \rangle \to_{\mathcal{P}_3} \langle \mathsf{q}(\mathsf{p}(\mathsf{b}(x)), \mathsf{p}(x)), \mathsf{p}(\mathsf{b}(x)) \rangle \xleftarrow{*}_{\mathcal{P}_3} \langle \mathsf{q}(\pi_1(\mathsf{r}(x)), \pi_2(\mathsf{r}(x))), \pi_1(\mathsf{r}(x)) \rangle \leftarrow_{\mathcal{P}_3} \mathsf{r1}(\mathsf{r}(x))$. Thus, $\mathcal{P}_3 \Rightarrow \mathcal{P}_4$ by the Addition rule.

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

7. Finally, applying the Elimination rule twice to $\mathcal{P}_5$, we obtain $\mathcal{P}'$.

Thus, $\langle \mathcal{P}, \mathcal{P}', \mathcal{H} \rangle$ is a correct template.

Example 4.10. The following TRS $\mathcal{R}_{\mathsf{fib}}$ computes Fibonacci number of input natural numbers. $\mathcal{R}_{\mathsf{fib}}$ is transformed to the TRS $\mathcal{R}'_{\mathsf{fib}}$ by the template $\langle \mathcal{P}, \mathcal{P}', \mathcal{H} \rangle$.

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

$$\mathcal{R}'_{\mathsf{fib}} \left\{ \begin{array}{lcl} \mathsf{fib}(\mathsf{0}) & \to & \mathsf{s}(\mathsf{0}) \\ \mathsf{fib}(\mathsf{s}(\mathsf{0})) & \to & \mathsf{s}(\mathsf{0}) \\ \mathsf{fib}(\mathsf{s}(\mathsf{s}(x))) & \to & \pi_1(\mathsf{f}_{\mathsf{r1}}(\mathsf{f}_{\mathsf{r}}(x))) \\ \mathsf{f}_{\mathsf{r}}(\mathsf{0}) & \to & \langle \mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{0}) \rangle \\ \mathsf{f}_{\mathsf{r}}(\mathsf{s}(x)) & \to & \mathsf{f}_{\mathsf{r1}}(\mathsf{f}_{\mathsf{r}}(x)) \\ \mathsf{f}_{\mathsf{r1}}(x) & \to & \langle +(\pi_1(x), \pi_2(x)), \pi_1(x) \rangle \end{array} \right. \quad \begin{array}{lcl} +(\mathsf{0}, x) & \to & x \\ +(\mathsf{s}(x), y) & \to & \mathsf{s}(+(x,y)) \\ \pi_1(\langle x, y \rangle) & \to & x \\ \pi_2(\langle x, y \rangle) & \to & y \end{array}$$

Let us consider another template. The following template $\langle \tilde{\mathcal{P}}, \tilde{\mathcal{P}}', \tilde{\mathcal{H}} \rangle$ deals with slightly more complicated tupling transformations.

$$\tilde{\mathcal{P}} \left\{ \begin{array}{lcl} \mathsf{p}(\mathsf{a}) & \to & \mathsf{b} \\ \mathsf{p}(\mathsf{c}(x,y)) & \to & \mathsf{h}(x, y, \mathsf{q}(y), \mathsf{p}(y)) \\ \mathsf{q}(\mathsf{a}) & \to & \mathsf{d} \\ \mathsf{q}(\mathsf{c}(x,y)) & \to & \mathsf{e}(x, y, \mathsf{q}(y)) \\ \pi_1(\langle x, y \rangle) & \to & x \\ \pi_2(\langle x, y \rangle) & \to & y \end{array} \right. \qquad \tilde{\mathcal{P}}' \left\{ \begin{array}{l} \mathsf{p}(\mathsf{a}) \to \mathsf{b} \\ \mathsf{p}(\mathsf{c}(x,y)) \to \pi_1(\mathsf{r}_1(x, y, \mathsf{r}(y))) \\ \mathsf{r}(\mathsf{a}) \to \langle \mathsf{b}, \mathsf{d} \rangle \\ \mathsf{r}(\mathsf{c}(x,y)) \to \mathsf{r1}(x, y, \mathsf{r}(y)) \\ \mathsf{r1}(x, y, z) \to \\ \qquad \langle \mathsf{h}(x, y, \pi_2(z), \pi_1(z)), \mathsf{e}(x, y, \pi_2(z)) \rangle \\ \mathsf{q}(\mathsf{a}) \to \mathsf{d} \\ \mathsf{q}(\mathsf{c}(x,y)) \to \mathsf{e}(x, y, \mathsf{q}(y)) \\ \pi_1(\langle x, y \rangle) \to x \\ \pi_2(\langle x, y \rangle) \to y \end{array} \right.$$

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

Here, π_1, π_2, and $\langle \cdot \rangle$ are function symbols.

We now show that there exists a correct transformation sequence from $\tilde{\mathcal{P}}$ to $\tilde{\mathcal{P}}'$ under $\tilde{\mathcal{H}}$, i.e. $\langle \tilde{\mathcal{P}}, \tilde{\mathcal{P}}', \tilde{\mathcal{H}} \rangle$ is a correct template.

1. Let $\mathcal{P}_0 = \tilde{\mathcal{P}}$.
2. Let $\mathcal{P}_1 = \mathcal{P}_0 \cup \{\mathsf{r}(x) \to \langle \mathsf{p}(x), \mathsf{q}(x) \rangle\}$. Here, r is a fresh pattern variable. Thus $\mathcal{P}_0 \Rightarrow \mathcal{P}_1$ by the Introduction rule.
3. Let $\mathcal{P}_2 = \mathcal{P}_1 \cup \{\mathsf{r1}(x, y, z) \to \langle \mathsf{h}(x, y, \pi_2(z), \pi_1(z)),\ \mathsf{e}(x, y, \pi_2(z)) \rangle\}$. Here, $\mathsf{r1}$ is a fresh pattern variable. Thus $\mathcal{P}_1 \Rightarrow \mathcal{P}_2$ by the Introduction rule.
4. Let $\mathcal{P}_3 = \mathcal{P}_2 \cup \{\mathsf{r}(\mathsf{a}) \to \langle \mathsf{b}, \mathsf{d} \rangle\}$. Here, we have $\mathsf{r}(\mathsf{a}) \to_{\mathcal{P}_2} \langle \mathsf{p}(\mathsf{a}), \mathsf{q}(\mathsf{a}) \rangle \xrightarrow{*}_{\mathcal{P}_2} \langle \mathsf{b}, \mathsf{d} \rangle$. Thus, $\mathcal{P}_2 \Rightarrow \mathcal{P}_3$ by the Addition rule.
5. Let $\mathcal{P}_4 = \mathcal{P}_3 \cup \{\mathsf{r}(\mathsf{c}(x,y)) \to \mathsf{r1}(x, y, \mathsf{r}(y))\}$. We have $\mathsf{r}(\mathsf{c}(x,y)) \to_{\mathcal{P}_3} \langle \mathsf{p}(\mathsf{c}(x,y)),\ \mathsf{q}(\mathsf{c}(x,y)) \rangle \xrightarrow{*}_{\mathcal{P}_3} \langle \mathsf{h}(x, y, \mathsf{q}(y), \mathsf{p}(y)), \mathsf{e}(x, y, \mathsf{q}(y)) \rangle \xleftarrow{*}_{\mathcal{P}_3} \langle \mathsf{h}(x, y, \pi_2(\mathsf{r}(y)), \pi_1(\mathsf{r}(y))), \mathsf{e}(x, y, \pi_2(\mathsf{r}(y))) \rangle \leftarrow_{\mathcal{P}_3} \mathsf{r1}(x, y, \mathsf{r}(y))$. Thus, $\mathcal{P}_3 \Rightarrow \mathcal{P}_4$ by the Addition rule.
6. Let $\mathcal{P}_5 = \mathcal{P}_4 \cup \{\mathsf{p}(\mathsf{c}(x,y)) \to \pi_1(\mathsf{r1}(x, y, \mathsf{r}(y)))\}$. Here, we have $\mathsf{p}(\mathsf{c}(x,y)) \leftarrow_{\mathcal{P}_4} \pi_1(\mathsf{r}(\mathsf{c}(x,y))) \to_{\mathcal{P}_4} \pi_1(\mathsf{r1}(x, y, \mathsf{r}(y)))$. Thus, $\mathcal{P}_4 \Rightarrow \mathcal{P}_5$ by the Addition rule.
7. Finally, applying the Elimination rule twice to $\mathcal{P}_5$, we obtain $\tilde{\mathcal{P}}'$.

Thus, $\langle \tilde{\mathcal{P}}, \tilde{\mathcal{P}}', \tilde{\mathcal{H}} \rangle$ is a correct template.

Example 4.11. A list of numbers is said to be *steep* if each element is greater than the sum of the elements that follow it [7]. The following TRS $\mathcal{R}_{\mathsf{steep}}$ specifies a program which checks whether input lists are steep. $\mathcal{R}_{\mathsf{steep}}$ is transformed to TRS $\mathcal{R}'_{\mathsf{steep}}$ by the template $\langle \tilde{\mathcal{P}}, \tilde{\mathcal{P}}', \tilde{\mathcal{H}} \rangle$.

$$\mathcal{R}_{\mathsf{steep}} \left\{ \begin{array}{lllllll} \mathsf{steep}(\mathsf{nil}) & \to & \mathsf{true} & \mathsf{and}(\mathsf{true},\mathsf{true}) & \to & \mathsf{true} \\ \mathsf{steep}(x{:}xs) & \to & & \mathsf{and}(\mathsf{true},\mathsf{false}) & \to & \mathsf{false} \\ \multicolumn{3}{r}{\mathsf{and}(\mathsf{gt}(x,\mathsf{sum}(xs)),\mathsf{steep}(xs))} & \mathsf{and}(\mathsf{false},\mathsf{true}) & \to & \mathsf{false} \\ \mathsf{sum}(\mathsf{nil}) & \to & 0 & \mathsf{and}(\mathsf{false},\mathsf{false}) & \to & \mathsf{false} \\ \mathsf{sum}(x{:}ys) & \to & +(x,\mathsf{sum}(ys)) & +(0,x) & \to & x \\ \mathsf{gt}(0,0) & \to & \mathsf{false} & +(\mathsf{s}(x),y) & \to & \mathsf{s}(+(x,y)) \\ \mathsf{gt}(0,\mathsf{s}(y)) & \to & \mathsf{false} & \pi_1(\langle x,y\rangle) & \to & x \\ \mathsf{gt}(\mathsf{s}(x),0) & \to & \mathsf{true} & \pi_2(\langle x,y\rangle) & \to & y \\ \mathsf{gt}(\mathsf{s}(x),\mathsf{s}(y)) & \to & \mathsf{gt}(x,y) & & & \end{array} \right.$$

$$\mathcal{R}'_{\mathsf{steep}} \left\{ \begin{array}{llllll} \mathsf{steep}([\,]) & \to & \mathsf{true} & \mathsf{gt}(\mathsf{s}(x),0) & \to & \mathsf{true} \\ \mathsf{steep}(x{:}y) & \to & \pi_1(\mathsf{f_{r1}}(x,y,\mathsf{f_r}(y))) & \mathsf{gt}(\mathsf{s}(x),\mathsf{s}(y)) & \to & \mathsf{gt}(x,y) \\ \mathsf{f_r}([\,]) & \to & \langle \mathsf{true}, 0\rangle & \mathsf{and}(\mathsf{false},\mathsf{false}) & \to & \mathsf{false} \\ \mathsf{f_r}(x{:}y) & \to & \mathsf{f_{r1}}(x,y,\mathsf{f_r}(y)) & \mathsf{and}(\mathsf{false},\mathsf{true}) & \to & \mathsf{false} \\ \mathsf{f_{r1}}(x,y,z) & \to & & \mathsf{and}(\mathsf{true},\mathsf{false}) & \to & \mathsf{false} \\ \multicolumn{3}{r}{\langle \mathsf{and}(\mathsf{gt}(x,\pi_2(z)),\pi_1(z)), +(x,\pi_2(z))\rangle} & \mathsf{and}(\mathsf{true},\mathsf{true}) & \to & \mathsf{true} \\ \mathsf{sum}([\,]) & \to & 0 & +(0,x) & \to & x \\ \mathsf{sum}(x{:}y) & \to & +(x,\mathsf{sum}(y)) & +(\mathsf{s}(x),y) & \to & \mathsf{s}(+(x,y)) \\ \mathsf{gt}(0,0) & \to & \mathsf{false} & \pi_1(\langle x,y\rangle) & \to & x \\ \mathsf{gt}(0,\mathsf{s}(y)) & \to & \mathsf{false} & \pi_2(\langle x,y\rangle) & \to & y \end{array} \right.$$

Example 4.12. The following TRS $\mathcal{R}_{\mathsf{factlist}}$ computes lists whose elements are factorial numbers.

$$\mathcal{R}_{\mathsf{factlist}} \left\{ \begin{array}{llllll} \mathsf{factlist}(0) & \to & [\,] & \times(0,y) & \to & 0 \\ \mathsf{factlist}(\mathsf{s}(x)) & \to & & \times(\mathsf{s}(x),y) & \to & +(y,\times(x,y)) \\ \multicolumn{3}{r}{\times(\mathsf{s}(x),\mathsf{fact}(x)){:}\mathsf{factlist}(x)} & +(0,x) & \to & x \\ \mathsf{fact}(0) & \to & \mathsf{s}(0) & +(\mathsf{s}(x),y) & \to & \mathsf{s}(+(x,y)) \\ \mathsf{fact}(\mathsf{s}(x)) & \to & \times(\mathsf{s}(x),\mathsf{fact}(x)) & \pi_1(\langle x,y\rangle) & \to & x \\ & & & \pi_2(\langle x,y\rangle) & \to & y \end{array} \right.$$

Since $\tilde{\mathcal{P}}$ can match to $\mathcal{R}_{\mathsf{factlist}}$, one might expect that $\mathcal{R}_{\mathsf{factlist}}$ is transformed by the template $\langle \tilde{\mathcal{P}}, \tilde{\mathcal{P}}', \tilde{\mathcal{H}} \rangle$. But this transformation fails. For, $\mathsf{factlist}(\mathsf{s}(y)) \to \pi_1(\mathsf{f_{r1}}(x,y,\mathsf{f_r}(y)))$ appears in $\varphi_{out}(\tilde{\mathcal{P}}')$, but this rule is not a rewrite rule (x appears in the rhs but the lhs). Indeed, φ cannot instantiate $\mathsf{r1}$ because φ has to carry the signature of $\tilde{\mathcal{P}}$ to $\mathcal{R}_{\mathsf{factlist}}$, that is, $\mathsf{r1} \notin \mathrm{dom}_{\mathscr{X}}(\varphi)$ must hold.

4.3 Summary

In this chapter, we gave a notion of equivalent transformation of TRSs which improved and simplified the technique proposed by Toyama[25]. We then proved that the equality of restricted TRSs is guaranteed by the equivalent transformation of TRSs (Theorem 4.2). The method to construct correct templates is given by lifting up the technique of equivalent transformation of TRSs to template level. We also introduced the notion of carrying signatures for term homomorphisms which preserve inferences of equivalent transformations. The definition of transformations by templates (Definition 4.5) takes account of carrying signatures for term homomorphisms. Theorem 4.9 showed that correct templates guarantee the correctness of transformations for restricted TRSs. We gave examples of correct templates and TRS transformations by them.

Chapter 5

Matching Algorithm

In this chapter, 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.

5.1 Term Pattern Matching

Definition 5.1. *(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 5.2. Suppose $\mathsf{f}, \mathsf{c} \in \mathscr{X}$ 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 5.3. *(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 5.1. 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 $\langle S, \emptyset\rangle \stackrel{*}{\Longrightarrow} \langle \emptyset, \varphi\rangle$.*

Table 5.1: Inference rules of **Match**

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**

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

Example 5.4. 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 5.1.

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

Figure 5.1: A sequence $\langle S, \emptyset\rangle \stackrel{*}{\Longrightarrow} \langle \emptyset, \varphi\rangle$ in the procedure **Match**

The next lemma is readily checked.

Lemma 5.5 (**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 5.6 (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 $[\langle|\mathscr{X}(s)|, |s|\rangle \mid s \trianglelefteq t \in S] >_m [\langle|\mathscr{X}(s)|, |s|\rangle \mid s \trianglelefteq t \in S']$ where $|\mathscr{X}(s)|$ 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. □

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

Theorem 5.7 (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. □

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 5.8. *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 5.9. *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 $\langle S, \varphi\rangle \Longrightarrow \langle S', \varphi'\rangle$. □

Theorem 5.10 (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 5.9, 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. □

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 5.11. *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 $\varphi \subseteq \varphi'$.*

Lemma 5.12. 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 $p \in \mathscr{X}$.

   Since $\tilde{\varphi}(p(s_1, \ldots, s_n)) = t$ and $t \in \mathrm{T}(\mathscr{F}, \mathscr{V})$, $p \in \mathrm{dom}_{\mathscr{X}}(\tilde{\varphi})$. 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 5.13 (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 5.10, any $\varphi \in \Phi$ is a matcher of S. Let $\tilde{\varphi}$ be a matcher of S. From Lemma 5.12, 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 5.6, this sequence is finite. So there exists φ' such that $\langle S, \emptyset\rangle \stackrel{*}{\Longrightarrow} \langle\emptyset, \varphi'\rangle$ and $\varphi' \subseteq \tilde{\varphi}$. Since $\langle S, \emptyset\rangle \stackrel{*}{\Longrightarrow} \langle\emptyset, \varphi'\rangle$ means $\varphi' \in \Phi$, the claim follows. □

5.2 TRS Pattern matching

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 5.14. *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 5.15. Let $\mathcal{P} \trianglelefteq \mathcal{R}$ be a TRS pattern matching problem where

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

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

This TRS pattern matching problem is encoded as a term pattern matching 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 5.16. *(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 5.17 (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 5.18 (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 5.10. (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 5.13, there exists a term homomorphism $\tilde{\varphi} \in \Phi'$ such that $\tilde{\varphi} \subseteq \varphi$ and $\tilde{\varphi}(S)$ is trivial. Thus for any $p, q \in \mathscr{X}_{\mathrm{d}}$, $\mathrm{root}(\tilde{\varphi}(p)) = \mathrm{root}(\tilde{\varphi}(q))$ implies $p = q$; for otherwise, φ is not a CS homomorphism by $\tilde{\varphi} \subseteq \varphi$. Since any defined pattern variable $p \in \mathscr{X}_{\mathrm{d}}$ appears at the root of some rewrite rule in $\mathcal{P}$, $p \in \mathrm{dom}_{\mathscr{X}}(\tilde{\varphi})$ holds. Thus for any $p \in \mathscr{X}_{\mathrm{d}}$, $\tilde{\varphi}(p) = \varphi(p) = f(\Box_{i_1}, \ldots, \Box_{i_n})$ for some $f \in \mathscr{F}_{\mathrm{d}}$. Therefore $\tilde{\varphi}$ is a CS homomorphism and hence $\tilde{\varphi} \in \Phi$. ☐

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 $p \in \mathscr{X} \backslash \mathrm{dom}_{\mathscr{X}}(\varphi)$ 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 5.19. Let $\mathcal{R}$ be a TRS in Example 3.1 and $\langle \mathcal{P}, \mathcal{P}', \mathcal{H} \rangle$ be a template in Example 3.3. 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.
$$\varphi = \left\{ \begin{array}{ll} \mathsf{f} \mapsto \mathsf{sum}(\Box_1), & u_1 \mapsto x_1, \\ \mathsf{g} \mapsto +(\Box_1, \Box_2), & v_1 \mapsto y_1, \\ \mathsf{a} \mapsto [\,], & u_2 \mapsto x_2, \\ \mathsf{b} \mapsto \mathsf{0}, & v_3 \mapsto x_3, \\ \mathsf{c} \mapsto \Box_1{:}\Box_2, & w_3 \mapsto y_3, \\ \mathsf{d} \mapsto \mathsf{s}(\Box_2) & \end{array} \right\}$$
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
$$\mathcal{R}' \quad \left\{ \begin{array}{lcl} \mathsf{sum}(u_4) & \to & \mathsf{sum1}(u_4, \mathsf{0}) \\ \mathsf{sum1}([\,], u_5) & \to & u_5 \\ \mathsf{sum1}(u_6{:}v_6, w_6) & \to & \mathsf{sum1}(v_6, +(w_6, u_6)) \\ +(\mathsf{0}, u_7) & \to & u_7 \\ +(\mathsf{s}(u_8), v_8) & \to & \mathsf{s}(+(u_8, v_8)) \end{array} \right.$$

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 [6, 11, 12] 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 [23] like this:

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

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.

5.3 Summary

In this chapter, we gave a term pattern matching algorithm **Match** and show its soundness and completeness (Theorem 5.10 and 5.13). We then proposed a TRS pattern matching algorithm **TRSMatch** by extending **Match**. We also compared the framework of program transformation by templates based on term rewriting and lambda calculus in the view of pattern matching.

Chapter 6

Program Transformation System RAPT

RAPT (Rewriting-based Automated Program Transformation system) is an implementation of our framework. This chapter describes about RAPT and reports experiments of transformations brought by RAPT. RAPT transforms input many-sorted TRSs according to specified correct templates and verifies its correctness automatically.

6.1 Implementation

A key property of our framework is sufficient completeness, which has to be satisfied by input and output TRSs. Sufficient completeness is checked in RAPT by the decidable necessary and sufficient condition for terminating TRSs [13, 17], and thus currently the target of program transformation by RAPT is limited to terminating TRSs. A simple procedure to check confluence is also available for terminating TRSs [1].

RAPT uses *rewriting induction* [21], in which termination plays an essential role, to verify that the instantiated hypotheses of transformation template are inductive consequences of the input TRS. Since RAPT handles only terminating TRSs, rewriting induction is integrated keeping the whole system simple. Other inductive proving methods [2, 4] also can be possibly incorporated.

For the termination checking, RAPT detects a possible compatible precedence for the lexicographic path ordering (LPO) [1]. The obtained reduction ordering is used as a basis of rewriting induction. Other methods to verify termination of TRSs [1] may well be incorporated.

6.1.1 Specification of input TRS and transformation template

Inputs of RAPT are a many-sorted TRS and a transformation template. The input TRS is specified by the following sections.

1. `FUNCTIONS`: function symbols with sort declaration.
2. `RULES`: rewrite rules over many-sorted terms.

The transformation template $\langle \mathcal{P}, \mathcal{P}', \mathcal{H} \rangle$ is specified by the following sections.

1. `INPUT`: rewrite rules of $\mathcal{P}$ over patterns,

```
FUNCTIONS                              INPUT
 sum:  List -> Nat;                     ?f(?a()) -> ?b();
 cons:  Nat * List -> List;             ?f(?c(u,v)) -> ?g(?e(u),?f(v));
 nil:  List;                            ?g(?b(),u) -> u;
 +:  Nat * Nat -> Nat;                  ?g(?d(u,v),w) -> ?d(u,?g(v,w))
 s:  Nat -> Nat;
 0:  Nat                               OUTPUT
                                        ?f(u) -> ?f1(u,?b());
RULES                                   ?f1(?a(),u) -> u;
 sum(nil()) -> 0();                     ?f1(?c(u,v),w) -> ?f1(v,?g(w,?e(u)));
 sum(cons(x,ys)) -> +(x,sum(ys));       ?g(?b(),u) -> u;
 +(0(), x) -> x;                        ?g(?d(u,v),w) -> ?d(u,?g(v,w))
 +(s(x),y) -> s(+(x,y))
                                        HYPOTHESIS
                                        ?g(?b(),u) = ?g(u,?b());
                                        ?g(?g(u,v),w) = ?g(u,?g(v,w))
```

Figure 6.1: Specification of input TRS and transformation template

2. `OUTPUT`: rewrite rules of $\mathcal{P}'$ over patterns,

3. `HYPOTHESIS`: equations of $\mathcal{H}$ over patterns.

Figure 6.1 shows the many-sorted TRS $\mathcal{R}_{sum}$ and the template $\langle \mathcal{P}, \mathcal{P}', \mathcal{H} \rangle$ which appear in Section 2 prepared as an input to RAPT: rules, equations and sort declarations are separated by "`;`"; pattern variables are preceded by "`?`"; and to distinguish variables from constants, the latter are followed by "`()`" .

6.1.2 Implementation details

RAPT is implemented using SML/NJ. The source code of RAPT consists of about 5,000 lines.

The TRS transformation and the verification of its correctness are conducted in RAPT in 6 phases. In Figure 6.2, we describe these phases and dependencies among each phase. Solid arrows represent data flow and dotted arrows explain how information obtained in each phase is used.

If these 6 phases are successfully passed then RAPT produces output TRSs. The correctness of the transformation is guaranteed, provided the transformation template is developed. RAPT can also report summaries of program transformation in a readable format (Figure 6.3).

We now explain operations of each phases briefly.

1. Validation of input TRS In this phase, RAPT checks whether the input TRS is left-linear and well-typed, and from rewrite rules divides function symbols into defined function symbols and constructor symbols and checks whether the input TRS is a constructor system. The information of function symbols will be used in Phases 3 and 4.

2. Precedence detection In this phase, RAPT checks the input TRS is terminating by LPO and (if it is the case) detects a precedence. The suitable precedence (if there exists one) for LPO is computed based on the LPO constraint solving algorithm described in [10].

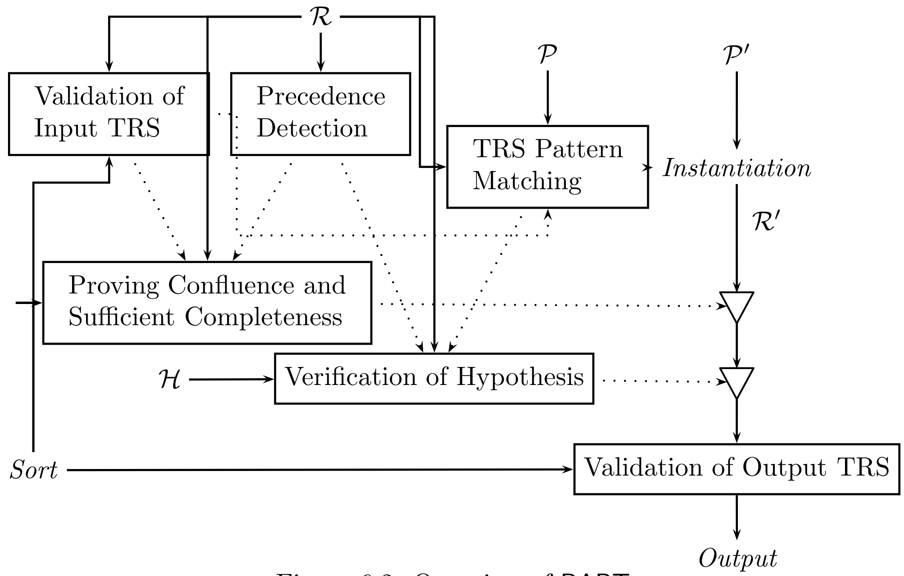


Figure 6.2: Overview of RAPT

3. Proving confluence and sufficient completeness In this phase, RAPT proves whether the input TRS is confluent and sufficiently complete. This makes use of the information of constructor symbols detected at Phase 1 and the fact that the input TRS is left-linear and terminating verified at Phases 1 and 2, respectively. For confluence, it is checked whether all critical pairs are joinable. For sufficient completeness, quasi-reducibility of the TRS is checked; this part is based on the (many-sorted extension of) complement algorithm introduced in [15] that computes the complement of a substitution.

4. TRS pattern matching In this phase, RAPT finds a combination of rewrite rules to apply the transformation and the term homomorphism which instantiates the input pattern TRS to these rewrite rules; the matching algorithm **TRSMatch** is used in this part. Using information of function symbols detected in Phase 1, it is also checked whether this term homomorphism is a CS-homomorphism. Pattern matching of rewrite rules are carried out in order, and use the information of matching solutions to limit next rewrite rules to perform the pattern match. Since solving the patten matching of main function usually gives information which subfunctions are used in sequel, this heuristics performs the TRS matching relatively well. Visually, consider the case when $\mathcal{P} = \{\mathsf{p}_i(x) \to \mathsf{p}_{i-1}(x) \mid 1 \leq i \leq 9\} \cup \{\mathsf{p}_0(x) \to \mathsf{a}\}$ and $\mathcal{R} = \{\mathsf{f}_i(x) \to \mathsf{f}_{i-1}(x) \mid 1 \leq i \leq 9\} \cup \{\mathsf{f}_0(x) \to 0\}$ where the number of all possible combinations of rewrite rules becomes $10! = 3,628,800$ while the number of matching performed becomes $\sum_{i=0}^{10} i = 55$.

5. Verification of hypothesis In this phase, RAPT checks whether the input TRS satisfies the hypothesis part of the template. This is done by (1) instantiating the hypotheses through the term homomorphism found at Phase 4 and (2) proving they are inductive consequences of the input TRS, using rewriting induction. The latter uses LPO with the precedence detected at Phase 2.

6. Validation of output TRS In this phase, RAPT checks whether the output TRS is (1) terminating, (2) left-linear, (3) type consistent, and (4) sufficiently complete. In (3), because the pattern TRS $\mathcal{P}'$ for the output may contain a pattern variable not occurring in the pattern TRS $\mathcal{P}$ for the input, types may be unknown for some of function symbols in $\mathcal{R}'$. Therefore,

Summary of Program Transformation

reported by RAPT
February 21, 2006

Transformation Template:

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

$$\mathcal{P}'\left\{\begin{array}{lcl} \mathsf{f}(u) & \to & \mathsf{f1}(u,\mathsf{b}) \\ \mathsf{f1}(\mathsf{a},u) & \to & u \\ \mathsf{f1}(\mathsf{c}(u,v),w) & \to & \mathsf{f1}(v,\mathsf{g}(w,\mathsf{e}(u,v))) \end{array}\right.$$

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

Input TRS:

$$\mathcal{R}\left\{\begin{array}{lcl} \mathsf{rev}(\mathsf{nil}) & \to & \mathsf{nil} \\ \mathsf{rev}(\mathsf{cons}(x,ys)) & \to & \mathsf{app}(\mathsf{rev}(ys),\mathsf{cons}(x,\mathsf{nil})) \\ \mathsf{app}(\mathsf{nil},x) & \to & x \\ \mathsf{app}(\mathsf{cons}(x,y),z) & \to & \mathsf{cons}(x,\mathsf{app}(y,z)) \end{array}\right.$$

Termination of $\mathcal{R}$ is checked by LPO with the precedence $\{\mathsf{rev} > \mathsf{app}, \mathsf{rev} > \mathsf{nil}, \mathsf{rev} > \mathsf{cons}, \mathsf{app} > \mathsf{cons}\}$. The set of critical pairs of $\mathcal{R}$ is $\{\}$.

A solution of matching (CS-homomorphisms):

$$\varphi = \left\{\begin{array}{lcl} \mathsf{b} & \mapsto & \mathsf{nil} \\ \mathsf{a} & \mapsto & \mathsf{nil} \\ \mathsf{e} & \mapsto & \mathsf{cons}(\Box_1,\mathsf{nil}) \\ \mathsf{g} & \mapsto & \mathsf{app}(\Box_2,\Box_1) \\ \mathsf{c} & \mapsto & \mathsf{cons}(\Box_1,\Box_2) \\ \mathsf{f} & \mapsto & \mathsf{rev}(\Box_1) \end{array}\right\}$$

The instantiation of hypothesis:

$$\varphi(\mathcal{H})\left\{\begin{array}{lcl} \mathsf{app}(u,\mathsf{nil}) & \approx & u \\ \mathsf{app}(\mathsf{nil},u) & \approx & u \\ \mathsf{app}(w,\mathsf{app}(v,u)) & \approx & \mathsf{app}(\mathsf{app}(w,v),u) \end{array}\right.$$

Output TRS:

$$\mathcal{R}'\left\{\begin{array}{lcl} \mathsf{rev}(u) & \to & \mathsf{f1}(u,\mathsf{nil}) \\ \mathsf{f1}(\mathsf{nil},u) & \to & u \\ \mathsf{f1}(\mathsf{cons}(u,v),w) & \to & \mathsf{f1}(v,\mathsf{cons}(u,w)) \\ \mathsf{app}(\mathsf{nil},x) & \to & x \\ \mathsf{app}(\mathsf{cons}(x,y),z) & \to & \mathsf{cons}(x,\mathsf{app}(y,z)) \end{array}\right.$$

Figure 6.3: Example of a program transformation report

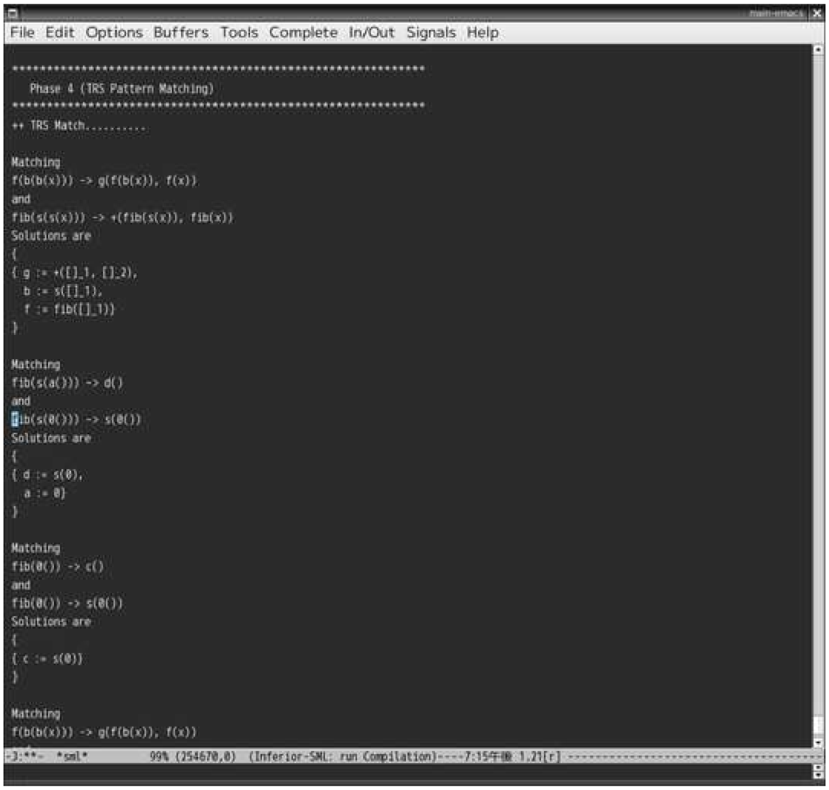


Figure 6.4: Snapshot of TRS pattern matching

we need to infer the type information together with the type consistency check. (4) is proved based on the fact the output TRS is terminating which is verified at (1) using LPO.

6.2 Experiments

We have checked operations of RAPT using several templates. Table 6.1 describes some of transformation templates and numbers of TRSs succeeded in transformation by each template. Template I is the one which appears in Section 2. This template represents a well-known transformation from recursive programs to iterative programs. A same kind of transformation is also described by Template II. The main difference between Template I and II is the right-hand side of second rule of input parts. In our experiments, there exist TRSs which cannot be transformed by one of these templates but can be done by the other. Template III is the one which overcomes this difference; unchanged rewrite rules of input and output TRS patterns are removed and rewrite rules which are necessary to develop the template are pushed into the hypothesis. Template IV represents another transformation known as fusion or deforestation [26].

RAPT performs transformations of these examples in less than 100 msec.

Table 6.1: Experimental result

Template I	TRSs	Template II	TRSs
$\left\{\begin{array}{lcl} \mathsf{f}(\mathsf{a}) & \rightarrow & \mathsf{b} \\ \mathsf{f}(\mathsf{c}(u,v)) & \rightarrow & \mathsf{g}(\mathsf{e}(u),\mathsf{f}(v)) \\ \mathsf{g}(\mathsf{b},u) & \rightarrow & u \\ \mathsf{g}(\mathsf{d}(u,v),w) & \rightarrow & \mathsf{d}(u,\mathsf{g}(v,w)) \end{array}\right\},$ $\left\{\begin{array}{lcl} \mathsf{f}(u) & \rightarrow & \mathsf{f}_1(u,\mathsf{b}) \\ \mathsf{f}_1(\mathsf{a},u) & \rightarrow & u \\ \mathsf{f}_1(\mathsf{c}(u,v),w) & \rightarrow & \mathsf{f}_1(v,\mathsf{g}(w,\mathsf{e}(u))) \\ \mathsf{g}(\mathsf{b},u) & \rightarrow & u \\ \mathsf{g}(\mathsf{d}(u,v),w) & \rightarrow & \mathsf{d}(u,\mathsf{g}(v,w)) \end{array}\right\},$ $\left\{\begin{array}{lcl} \mathsf{g}(\mathsf{b},u) & \approx & \mathsf{g}(u,\mathsf{b}) \\ \mathsf{g}(\mathsf{g}(u,v),w) & \approx & \mathsf{g}(u,\mathsf{g}(v,w)) \end{array}\right\}$	3	$\left\{\begin{array}{lcl} \mathsf{f}(\mathsf{a}) & \rightarrow & \mathsf{b} \\ \mathsf{f}(\mathsf{c}(u,v)) & \rightarrow & \mathsf{g}(\mathsf{f}(v),\mathsf{e}(u)) \\ \mathsf{g}(\mathsf{b},u) & \rightarrow & u \\ \mathsf{g}(\mathsf{d}(u,v),w) & \rightarrow & \mathsf{d}(u,\mathsf{g}(v,w)) \end{array}\right\},$ $\left\{\begin{array}{lcl} \mathsf{f}(u) & \rightarrow & \mathsf{f}_1(u,\mathsf{b}) \\ \mathsf{f}_1(\mathsf{a},u) & \rightarrow & u \\ \mathsf{f}_1(\mathsf{c}(u,v),w) & \rightarrow & \mathsf{f}_1(v,\mathsf{g}(\mathsf{e}(u),w)) \\ \mathsf{g}(\mathsf{b},u) & \rightarrow & u \\ \mathsf{g}(\mathsf{d}(u,v),w) & \rightarrow & \mathsf{d}(u,\mathsf{g}(v,w)) \end{array}\right\},$ $\left\{\begin{array}{lcl} \mathsf{g}(\mathsf{b},u) & \approx & \mathsf{g}(u,\mathsf{b}) \\ \mathsf{g}(\mathsf{g}(u,v),w) & \approx & \mathsf{g}(u,\mathsf{g}(v,w)) \end{array}\right\}$	3

Template III	TRSs	Template IV	TRSs
$\left\{\begin{array}{lcl} \mathsf{f}(\mathsf{a}) & \rightarrow & \mathsf{b} \\ \mathsf{f}(\mathsf{c}(u,v)) & \rightarrow & \mathsf{g}(\mathsf{e}(u,v),\mathsf{f}(v)) \end{array}\right\},$ $\left\{\begin{array}{lcl} \mathsf{f}(u) & \rightarrow & \mathsf{f}_1(u,\mathsf{b}) \\ \mathsf{f}_1(\mathsf{a},u) & \rightarrow & u \\ \mathsf{f}_1(\mathsf{c}(u,v),w) & \rightarrow & \mathsf{f}_1(v,\mathsf{g}(w,\mathsf{e}(u,v))) \end{array}\right\},$ $\left\{\begin{array}{lcl} \mathsf{g}(\mathsf{b},u) & \approx & u \\ \mathsf{g}(u,\mathsf{b}) & \approx & u \\ \mathsf{g}(\mathsf{g}(u,v),w) & \approx & \mathsf{g}(u,\mathsf{g}(v,w)) \end{array}\right\}$	11	$\left\{\begin{array}{lcl} \mathsf{f}(x,y,z) & \rightarrow & \mathsf{g}(\mathsf{h}(x,y),z) \\ \mathsf{g}(\mathsf{a},y) & \rightarrow & \mathsf{b}(u) \\ \mathsf{g}(\mathsf{c}(x,y),z) & \rightarrow & \mathsf{e}(x,\mathsf{g}(y,z)) \\ \mathsf{h}(\mathsf{a},y) & \rightarrow & \mathsf{r}(y) \\ \mathsf{h}(\mathsf{c}(x,y),z) & \rightarrow & \mathsf{c}(\mathsf{d}(x),\mathsf{h}(y,z)) \end{array}\right\},$ $\left\{\begin{array}{lcl} \mathsf{f}(\mathsf{a},y,z) & \rightarrow & \mathsf{g}(\mathsf{r}(y),z) \\ \mathsf{f}(\mathsf{c}(x,y),z,w) & \rightarrow & \mathsf{e}(\mathsf{d}(x),\mathsf{f}(y,z,w)) \\ \mathsf{g}(\mathsf{a},y) & \rightarrow & \mathsf{b}(u) \\ \mathsf{g}(\mathsf{c}(x,y),z) & \rightarrow & \mathsf{e}(x,\mathsf{g}(y,z)) \\ \mathsf{h}(\mathsf{a},y) & \rightarrow & \mathsf{r}(y) \\ \mathsf{h}(\mathsf{c}(x,y),z) & \rightarrow & \mathsf{c}(\mathsf{d}(x),\mathsf{h}(y,z)) \end{array}\right\},$ $\{\}$	8

Chapter 7

Constructing Templates

To apply the technique of program transformation by template, appropriate transformation patterns have to be constructed beforehand. Thus, it is important to introduce new transformation patterns in order to enhance the variety of program transformation. Up to our knowledge, however, few works discuss about the construction of transformation templates.

Our idea is to construct transformation patterns by considering the opposite of problems of program transformation, that is, we try to construct transformation patterns by generalizing similar TRS transformations. For example, from TRS transformations $\mathcal{R}_{sum} \Rightarrow \mathcal{R}'_{sum}$ and $\mathcal{R}_{cat} \Rightarrow \mathcal{R}'_{cat}$, we try to construct the transformation pattern $\mathcal{P} \Rightarrow \mathcal{P}'$. We expect that our method will help to extract new transformation patterns from existing program transformations.

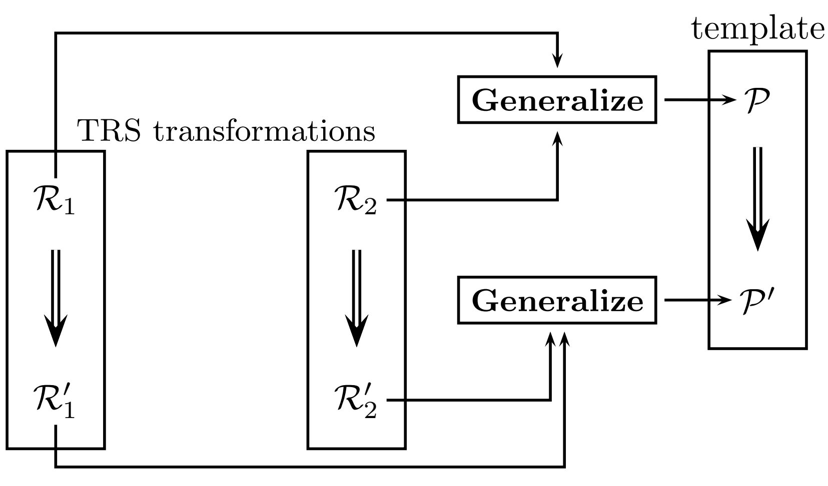


Figure 7.1: Overview of the construction of a template

We first propose a generalization procedure of two terms, and extend it for two TRSs. We then propose the construction of transformation patterns using the generalization procedure of TRSs. The input part of the transformation pattern is constructed by generalizing inputs of program transformations. Then the output part is constructed by generalizing outputs of program transformations using the information of generalization of input part. (Fig. 7.1).

Our method is inspired by Plotkin's work[20] for the *first-order generalization* of terms. The key technique of our method is the *2nd-order generalization* of terms; contrast to the first-order generalization, a function part of a term can be instantiated in the 2nd-order generalization. For example, a first-order generalization of $+(\mathsf{s}(x_1), y_1)$ and $+(x_2, \mathsf{s}(y_2))$ is $+(x_3, y_3)$. On the other hand, a 2nd-order generalization of $+(\mathsf{s}(x_1), y_1)$ and $\times(\mathsf{s}(x_2), y_2)$ is $\mathsf{p}(\mathsf{s}(x_3), y_3)$ where p is a pattern variable that is instantiated by $+$ or $\times$.

An important problem in program transformation is to guarantee its correctness. We say that a program transformation is correct when the input and output program perform the same computation. In fact, incorrect transformations may be also obtained by the transformation pattern $\mathcal{P} \Rightarrow \mathcal{P}'$ above. We have defined a transformation template by a triple $\langle \mathcal{P}, \mathcal{P}', \mathcal{H} \rangle$ where $\mathcal{P}$ and $\mathcal{P}'$ are used to form the transformation pattern $\mathcal{P} \Rightarrow \mathcal{P}'$ and $\mathcal{H}$, called hypothesis, is a set of equations. A hypothesis $\mathcal{H}$ is used to represent lemmas which input TRSs have to satisfy to guarantee the correctness of transformation.

Currently, no automatic method to produce correct templates is known. In our framework, after constructing a transformation pattern by generalizing input similar transformations, we look for an appropriate hypothesis and prove the correctness to construct correct template (Fig. 7.1).

7.1 Generalization of Terms

In this section, we propose a term generalization procedure, called **2nd-Gen**, and show its soundness. **2nd-Gen** will be used as a basic module of TRS generalization procedure. We first give a notion of generalization of two term patterns.

Definition 7.1. Let s and t be term patterns. A term pattern u is a *generalization of s and t* if there exist term homomorphisms φ_1 and φ_2 such that $\varphi_1(u) = s$ and $\varphi_2(u) = t$.

Example 7.2. Let $\mathsf{f}, \mathsf{g} \in \mathscr{F}$, $\mathsf{p}, \mathsf{q} \in \mathscr{X}$ and $x, y, z \in \mathscr{V}$. Then

1. $\mathsf{p}(x, y)$ is a generalization of $\mathsf{f}(x, x)$ and $\mathsf{g}(y)$, since $\varphi_1(\mathsf{p}(x, y))) = \mathsf{f}(x, x)$ and $\varphi_2(\mathsf{p}(x, y)) = \mathsf{g}(y)$ for $\varphi_1 = \{\mathsf{p} \mapsto \mathsf{f}(\Box_1, \Box_1)\}$, $\varphi_2 = \{\mathsf{p} \mapsto \mathsf{g}(\Box_2)\}$.

2. $\mathsf{p}(z)$ is a generalization of $\mathsf{f}(x, x)$ and $\mathsf{g}(y)$, since $\varphi_1(\mathsf{p}(z))) = \mathsf{f}(x, x)$ and $\varphi_2(\mathsf{p}(z)) = \mathsf{g}(y)$ for $\varphi_1 = \{\mathsf{p} \mapsto \mathsf{f}(\Box_1, \Box_1), z \mapsto x\}$, $\varphi_2 = \{\mathsf{p} \mapsto \mathsf{g}(\Box_1), z \mapsto y\}$.

3. $\mathsf{p}(\mathsf{q}(z))$ is a generalization of $\mathsf{f}(x, x)$ and $\mathsf{g}(y)$, since $\varphi_1(\mathsf{p}(\mathsf{q}(z)))) = \mathsf{f}(x, x)$ and $\varphi_2(\mathsf{p}(\mathsf{q}(z))) = \mathsf{g}(y)$ for $\varphi_1 = \{\mathsf{p} \mapsto \mathsf{f}(\Box_1, \Box_1), \mathsf{q} \mapsto \Box_1, z \mapsto x\}$, $\varphi_2 = \{\mathsf{p} \mapsto \Box_1, \mathsf{q} \mapsto \mathsf{g}(\Box_1), z \mapsto y\}$.

Our generalization procedure **2nd-Gen** given later computes a generalization of two input term patterns in a non-deterministic way. Table 7.1 explains how two input term patterns $\mathsf{f}(\mathsf{g}(x), y)$ and $\mathsf{f}(z, \mathsf{h}(u, w))$ are generalized into $\mathsf{f}(\mathsf{p}(v_1), \mathsf{q}(v_2, u))$ using **2nd-Gen**.

Initially, two input terms $\mathsf{f}(\mathsf{g}(x), y)$ and $\mathsf{f}(z, \mathsf{h}(u, w))$ are coupled into $\mathsf{f}(\mathsf{g}(x), y) \wedge \mathsf{f}(z, \mathsf{h}(u, w))$, using a special binary function symbol $\wedge$ (step 1). Since $\wedge$ indicates the position which will be generalized, nesting of $\wedge$ is not allowed. Next, **2nd-Gen** repeats the following process depending on two symbols α and β immediately below some $\wedge$, until it obtains a solution.

I If α and β are local variables, then the coupled local variables $\alpha \wedge \beta$ is replaced with a new local variable. The *memorizing function* records the association between the coupled local variables and the introduced local variable.

II If α and β are the same function symbols or pattern variables, then the symbol $\wedge$ is distributed in each argument.

III Otherwise, the coupled contexts is replaced with a new pattern variable and the modified arguments. The *memorizing function* records the association between the coupled contexts and the introduced pattern variable.

Var

$$\frac{C[x \wedge y],\ \Phi}{C[z]\theta,\ \Phi \cup \{x \wedge y \mapsto z\}} \quad \begin{array}{l}\text{if either} \\ \quad (1)\ \Phi(x \wedge y) = z \text{ or} \\ \quad (2)\ x \notin \mathrm{range}(\Phi_{[1]}^{-1}),\ y \notin \mathrm{range}(\Phi_{[2]}^{-1}), \text{ and } z \text{ is a fresh local variable} \\ \text{where } \theta = \{x := z, y := z\} \text{ is a substitution.}\end{array}$$

Div

$$\frac{C[p(s_1, \ldots, s_n) \wedge p(t_1, \ldots, t_n)],\ \Phi}{C[p(s_1 \wedge t_1, \ldots, s_n \wedge t_n)],\ \Phi} \qquad \text{if } p \in \mathscr{F} \cup \mathscr{X}$$

Gen

$$\frac{C[C_1\langle s_1, \ldots, s_n\rangle \wedge C_2\langle t_1, \ldots, t_n\rangle],\ \Phi}{C[p(\alpha_1, \ldots, \alpha_n)],\ \Phi \cup \{C_1 \wedge C_2 \mapsto p\}} \quad \begin{array}{l}\text{if either } \Phi(C_1 \wedge C_2) = p \text{ or} \\ \quad (1)\ C_1, C_2 \in T_n^{\square}(\mathscr{F} \cup \mathscr{X}), C_1 \neq C_2, \\ \quad (2)\ p \text{ is a fresh } (n\text{-ary}) \text{ pattern variable} \\ \quad (3)\ C_1 \wedge C_2 \notin \mathrm{dom}(\Phi) \\ \quad (4)\ \mathscr{H}(C_1) \cup \mathscr{H}(C_2) = \{\square_1, \ldots, \square_n\}, \text{ and} \\ \quad (5)\ \alpha_i = \begin{cases} s_i \wedge t_i & \text{if } \square_i \in \mathscr{H}(C_1) \cap \mathscr{H}(C_2) \\ \Phi_{[1]}(s_i) & \text{if } \square_i \in \mathscr{H}(C_1) \setminus \mathscr{H}(C_2) \\ \Phi_{[2]}(t_i) & \text{if } \square_i \in \mathscr{H}(C_2) \setminus \mathscr{H}(C_1) \end{cases}\end{array}$$

Figure 7.2: Inference rules of **2nd-Gen**

Let $\wedge$ be a special binary function symbol. A coupled term pattern is defined as follows.

Definition 7.3. The set $\mathrm{T}\wedge(\mathscr{F} \cup \mathscr{X}, \mathscr{V})$ of *coupled term patterns* is defined as follow: (i) $\mathrm{T}(\mathscr{F} \cup \mathscr{X}, \mathscr{V}) \subseteq \mathrm{T}\wedge(\mathscr{F} \cup \mathscr{X}, \mathscr{V})$; (ii) $s, t \in \mathrm{T}(\mathscr{F} \cup \mathscr{X}, \mathscr{V})$ implies $s \wedge t \in \mathrm{T}\wedge(\mathscr{F} \cup \mathscr{X}, \mathscr{V})$; (iii) if $s_1, \ldots, s_n \in \mathrm{T}\wedge(\mathscr{F} \cup \mathscr{X}, \mathscr{V})$, $p \in \mathscr{F} \cup \mathscr{X}$ and $\mathrm{arity}(p) = n$ then $p(s_1, \ldots, s_n) \in \mathrm{T}\wedge(\mathscr{F} \cup \mathscr{X}, \mathscr{V})$.

From the definition it is clear that every coupled term patten has no nested $\wedge$ symbols. A coupled term pattern t is $\wedge$*-free* if $t \in \mathrm{T}(\mathscr{F} \cup \mathscr{X}, \mathscr{V})$. A coupled term pattern t is $\wedge$*-top* if $t = t' \wedge t''$ for some $t', t'' \in \mathrm{T}(\mathscr{F} \cup \mathscr{X}, \mathscr{V})$.

Each term homomorphism φ and each substitution θ are extended to coupled term patterns by $\varphi(s \wedge t) = \varphi(s) \wedge \varphi(t)$ and $\theta(s \wedge t) = s \wedge t$ respectively. Note that the symbol $\wedge$ cancels the substitution to the term patterns below it (i.e. $\theta(s \wedge t) \neq \theta(s) \wedge \theta(t)$ in general). The set $\mathrm{T}\wedge(\mathscr{F} \cup \mathscr{X} \cup \mathscr{H}, \mathscr{V})$ is defined similarly.

Definition 7.4. Let t be a coupled term pattern. For $i = 1, 2$, the (first and second) *projection* $\pi_i(t)$ of t is defined as follows:

$$\pi_i(t) = \begin{cases} t & \text{if } t \in \mathrm{T}(\mathscr{F} \cup \mathscr{X}, \mathscr{V}) \\ p(\pi_i(s_1), \ldots, \pi_i(s_n)) & \\ \quad \text{if } t = p(s_1, \ldots, s_n) \text{ for } p \in \mathscr{F} \cup \mathscr{X} & \\ s_i & \text{if } t = s_1 \wedge s_2 \end{cases}$$

Example 7.5. *Let* $\mathsf{f}, \mathsf{g} \in \mathscr{F}$ *and* $x, y \in \mathscr{V}$. *Then* $s_1 = \mathsf{f}(x, x) \wedge \mathsf{g}(y)$, $s_2 = \mathsf{f}(x \wedge y, x)$, $s_3 = \mathsf{f}(x \wedge y, x \wedge \mathsf{g}(y))$ *are coupled term patterns but* $\mathsf{f}(x \wedge (x \wedge y), x)$ *is not because it has nested* $\wedge$ *symbols. The* $\wedge$*-top subterms of* s_3 *are* $x \wedge y$ *and* $x \wedge \mathsf{g}(y)$. *Also, we have* $\pi_1(s_1) = \pi_1(s_2) = \pi_1(s_3) = \mathsf{f}(x, x)$, $\pi_2(s_1) = \mathsf{g}(y)$, $\pi_2(s_2) = \mathsf{f}(y, x)$, *and* $\pi_2(s_3) = \mathsf{f}(y, \mathsf{g}(y))$.

From the definition the following properties of the projection are obtained easily.

Lemma 7.6. *Let $i = 1$ or 2.*

1. *If s is $\wedge$-free then $\pi_i(s) = s$.*
2. *For any term homomorphism φ and coupled term pattern s, $\pi_i(\varphi(s)) = \varphi(\pi_i(s))$.*
3. *For any coupled term pattern $C[s_1 \wedge s_2]$, $\pi_i(C[s_1 \wedge s_2]) = \pi_i(C[s_i])$.*

The memorizing function Φ, which records the association between the coupled contexts (the coupled local variables) and the introduced pattern variables (the introduced local variables, respectively), is carried along with the coupled term pattern during the generalization.

Definition 7.7. A *memorizing function* is a partial mapping Φ from $\{C_1 \wedge C_2 \mid C_1, C_2 \in \mathrm{T}^{\square}(\mathscr{F} \cup \mathscr{X})\} \cup \{x \wedge y \mid x, y \in \mathscr{V}\}$ to $\mathscr{X} \cup \mathscr{V}$ such that (1) $\Phi(x \wedge y) \in \mathscr{V}$ and $\Phi(C_1 \wedge C_2) \in \mathscr{X}$, (2) $\Phi(x \wedge y)$ and $\Phi(C_1 \wedge C_2)$ are fresh local variables and pattern variables (i.e., different from all the variables already used), respectively, (3) $x \wedge y, x \wedge y' \in \mathrm{dom}(\Phi)$ (or $y \wedge x, y' \wedge x \in \mathrm{dom}(\Phi)$) implies $y = y'$, (4) If $C_1 \wedge C_2 \mapsto p \in \Phi$ and $\mathrm{arity}(p) = n$, then $C_1 \neq C_2$, $C_1, C_2 \in T_n^{\square}(\mathscr{F} \cup \mathscr{X})$, and $\mathscr{H}(C_1) \cup \mathscr{H}(C_2) = \{\square_1, \ldots, \square_n\}$.

For a memorizing function Φ, its inverse projection is a term homomorphism defined by $\Phi_{[i]}^{-1} = \{u \mapsto s_i \mid s_1 \wedge s_2 \mapsto u \in \Phi\}$, and its local projection is a substitution defined by $\Phi_{[i]} = \{x_i := z \mid x_1 \wedge x_2 \mapsto z \in \Phi, z \in \mathscr{V}\}$. From the condition (3) of the memorizing function, the local projection $\Phi_{[i]}$ is well-defined.

The memorization function has the next property which follows immediately from the definition.

Lemma 7.8. *Let Φ be a memorizing function. Let s be a $\wedge$-free term such that $\mathscr{V}(s) \cap \mathrm{range}(\Phi) = \emptyset$. Then $\Phi_{[i]}^{-1}(\Phi_{[i]}(s)) = s$.*

The generalization procedure **2nd-Gen** works on pairs $\langle s, \Phi \rangle$ of a coupled term pattern s and a memorizing function Φ. Figure 7.2 gives the inference rules of **2nd-Gen**. For pairs $\langle s, \Phi \rangle$ and $\langle s', \Phi' \rangle$, we write $\langle s, \Phi \rangle \rightsquigarrow \langle s', \Phi' \rangle$ when $\langle s', \Phi' \rangle$ is obtained from $\langle s, \Phi \rangle$ by applying one of the inference rules in Figure 7.2. The reflexive transitive closure of $\rightsquigarrow$ is denoted by $\overset{*}{\rightsquigarrow}$.

The generalization procedure **2nd-Gen** is given as follows:

procedure **2nd-Gen**
Input: term patterns s and t
begin
1. Rename local variables of s and t so that $\mathscr{V}(s)$ and $\mathscr{V}(t)$ are disjoint.
2. Compute $\langle s \wedge t, \emptyset \rangle \overset{*}{\rightsquigarrow} \langle u, \Phi \rangle$ until u becomes $\wedge$-free.
3. Output a term pattern u

end.

Since there exist several possibilities for applying the rule **Gen**, two input term patterns s and t may have more than one generalization. For example, $\mathsf{p}(u, u)$ and $\mathsf{q}(\mathsf{h}, v)$ are generalizations of $\mathsf{f}(\mathsf{a}, x)$ and $\mathsf{g}(y, y)$. We note that for a given coupled term pattern the number of possible combinations of C_1 and C_2 in the rule **Gen** is finite, because of the condition (4) of **Gen**.

Lemma 7.9. *The procedure **2nd-Gen** is well-defined.*

Proof. It suffices to show that if Φ is a memorizing function and $\langle s, \Phi\rangle \rightsquigarrow \langle s', \Phi'\rangle$ then Φ' is again a memorizing function. We distinguish cases by the inference rule applied in the step $\langle s, \Phi\rangle \rightsquigarrow \langle s', \Phi'\rangle$.

(Var) The case $x \wedge y \mapsto z \in \Phi$ is obvious. Suppose $x \wedge y \mapsto z \notin \Phi$. Then $\Phi' = \Phi \cup \{x \wedge y \mapsto z\}$, $x \notin \mathrm{range}(\Phi^{-1}_{[1]})$, $y \notin \mathrm{range}(\Phi^{-1}_{[2]})$, and z is a fresh local variable. Clearly, Φ' is a partial mapping from $\{C_1 \wedge C_2 \mid C_1, C_2 \in \mathrm{T}^{\Box}(\mathscr{F} \cup \mathscr{X})\} \cup \{x \wedge y \mid x, y \in \mathscr{V}\}$ to $\mathscr{X} \cup \mathscr{V}$. The conditions (1),(2),(4) are clearly satisfied. The condition (3) follows since $x \notin \mathrm{range}(\Phi^{-1}_{[1]})$ and $y \notin \mathrm{range}(\Phi^{-1}_{[2]})$.

(Div) Since $\Phi' = \Phi$, the claim follows immediately.

(Gen) The case $C_1 \wedge C_2 \mapsto p \in \Phi$ is obvious. So, suppose $C_1 \wedge C_2 \mapsto p \notin \Phi$. By $C_1, C_2 \in \mathrm{T}^{\Box}(\mathscr{F} \cup \mathscr{X})$, Φ' is a partial mapping $\{C_1 \wedge C_2 \mid C_1, C_2 \in \mathrm{T}^{\Box}(\mathscr{F} \cup \mathscr{X})\} \cup \{x \wedge y \mid x, y \in \mathscr{V}\}$ to $\mathscr{X} \cup \mathscr{V}$. It is easy to check the conditions (1),(2),(3),(4) are satisfied.

□

Example 7.10. *We present some examples of the derivation of* ***2nd-Gen****. Recall that the symbol* $\wedge$ *cancels the substitution* θ*, that is,* $\theta(s \wedge t) = s \wedge t$*.*

1. $\langle f(x,x,x) \wedge g(y,y), \emptyset\rangle \rightsquigarrow_{\mathbf{Gen}} \langle p(x \wedge y, x, x \wedge y), \{f(\Box_1, \Box_2, \Box_3) \wedge g(\Box_1, \Box_3) \mapsto p\}\rangle \rightsquigarrow_{\mathbf{Var}} \langle p(z, z, x \wedge y), \{f(\Box_1, \Box_2, \Box_3) \wedge g(\Box_1, \Box_3) \mapsto p, x \wedge y \mapsto z\}\rangle \rightsquigarrow_{\mathbf{Var}} \langle p(z,z,z), \{f(\Box_1, \Box_2, \Box_3) \wedge g(\Box_1, \Box_3) \mapsto p, x \wedge y \mapsto z\}\rangle$.
2. $\langle f(x, h(x)) \wedge f(y, g(y)), \emptyset\rangle \rightsquigarrow_{\mathbf{Div}} \langle f(x \wedge y, h(x) \wedge g(y)), \emptyset\rangle \rightsquigarrow_{\mathbf{Var}} \langle f(z, h(x) \wedge g(y)), \{x \wedge y \mapsto z\}\rangle \rightsquigarrow_{\mathbf{Gen}} \langle f(z, q(x \wedge y)), \{x \wedge y \mapsto z, h(\Box_1) \wedge g(\Box_1) \mapsto q\}\rangle \rightsquigarrow_{\mathbf{Var}} \langle f(z, q(z)), \{x \wedge y \mapsto z, h(\Box_1) \wedge g(\Box_1) \mapsto q\}\rangle$.

We next show that the procedure **2nd-Gen** eventually terminates for any input, by using the following measure.

Definition 7.11. For $t \in \mathrm{T}\wedge(\mathscr{F} \cup \mathscr{X}, \mathscr{V})$, the *weight* $w(t)$ of a coupled term pattern t is a multiset of natural numbers defined as follows:

$$w(t) = \begin{cases} [\,] & \text{if } t \in \mathrm{T}(\mathscr{F} \cup \mathscr{X}, \mathscr{V}) \\ \bigsqcup_{i=1}^{n} w(s_i) & \text{if } t = p(s_1, \ldots, s_n) \\ & \text{with } p \in \mathscr{F} \cup \mathscr{X} \\ [\,|s_1| + |s_2|\,] & \text{if } t = s_1 \wedge s_2 \end{cases}$$

where $|s|$ denotes the number of symbol occurrences.

Theorem 7.12. The procedure **2nd-Gen** terminates for any input.

Proof. It suffices to show $\rightsquigarrow$ is well-founded. Thus, we prove that $\langle s, \Phi\rangle \rightsquigarrow \langle s', \Phi'\rangle$ implies $w(s) \gg w(s')$ where $\gg$ is the multiset extension of $>$[1]. We distinguish cases by the inference rule applied in the step $\langle s, \Phi\rangle \rightsquigarrow \langle s', \Phi'\rangle$.

(Var) One occurrence of $x \wedge y$ is replaced by z, and thus $w(s) = w(s') \sqcup [2]$. Hence $w(s) \gg w(s')$.

(Div) One occurrence of $p(s_1, \ldots, s_n) \wedge p(t_1, \ldots, t_n)$ is replaced by $p(s_1 \wedge t_1, \ldots, s_n \wedge t_n)$. Since $|p(s_1, \ldots, s_n) \wedge p(t_1, \ldots, t_n)| = |s_1| + \cdots + |s_n| + |t_1| + \cdots + |t_n| + 2$ and $[|s_1 \wedge t_1|, \ldots, |s_n \wedge t_n|] = [|s_1| + |t_1|, \ldots, |s_n| + |t_n|]$, we have $w(s) \gg w(s')$.

(Gen) In this case, we have $w(p(\alpha_1,\ldots,\alpha_n)) = [|s_i| + |t_i| \mid \square_i \in \mathscr{H}(C_1) \cap \mathscr{H}(C_2)]$ and $w(C_1\langle s_1,\ldots,s_n\rangle \wedge C_2\langle t_1,\ldots,t_n\rangle) = [|C_1\langle s_1,\ldots,s_n\rangle| + |C_2\langle t_1,\ldots,t_n\rangle|]$. Since $\square_i \in \mathscr{H}(C_1) \cap \mathscr{H}(C_2)$ implies $s_i \trianglelefteq C_1\langle s_1,\ldots,s_n\rangle$ and $t_i \trianglelefteq C_2\langle t_1,\ldots,t_n\rangle$, $|C_1\langle s_1,\ldots,s_n\rangle| + |C_2\langle t_1,\ldots,t_n\rangle| \geq |s_i| + |t_i|$ for i such that $\square_i \in \mathscr{H}(C_1) \cap \mathscr{H}(C_2)$. Thus the case $s_i \neq C_1\langle s_1,\ldots,s_n\rangle$ or $t_i \neq C_2\langle t_1,\ldots,t_n\rangle$ follows clearly. If $s_i = C_1\langle s_1,\ldots,s_n\rangle$ and $t_i = C_2\langle t_1,\ldots,t_n\rangle$ then $C_1 = \square_1 = C_2$, thus this case does not happen by the condition of the inference rule.

$\square$

Now we show the soundness of the procedure **2nd-Gen**, that is, every output of **2nd-Gen** is a generalization of two input term patterns. The following lemma is shown easily.

Lemma 7.13. *For any indexed context C such that $\square_i \notin \mathscr{H}(C)$ and any term patterns $s_1,\ldots,s_n,t_i$, $C\langle s_1,\ldots,s_i,\ldots,s_n\rangle = C\langle s_1,\ldots,t_i,\ldots,s_n\rangle$.*

We now prove the main lemma for the soundness theorem.

Lemma 7.14. *Let $\langle s,\Phi\rangle \rightsquigarrow \langle s',\Phi'\rangle$. Let $\mathscr{V}_1$ and $\mathscr{V}_2$ be disjoint sets of local variables. Suppose that, for $i \in \{1,2\}$, (1) $\mathscr{V}(\Phi^{-1}_{[i]}(\pi_i(s))) \subseteq \mathscr{V}_i$ and (2) for any $\wedge$-top subterm $u_1 \wedge u_2$ of s, $\mathscr{V}(u_i) \subseteq \mathscr{V}_i$. Then, for each $i \in \{1,2\}$, $\Phi^{-1}_{[i]}(\pi_i(s)) = \Phi'^{-1}_{[i]}(\pi_i(s'))$. Also, conditions (1) and (2) hold for Φ' and s'.*

Proof. We distinguish cases by the inference rule applied in the step $\langle s,\Phi\rangle \rightsquigarrow \langle s',\Phi'\rangle$. We show only $\Phi^{-1}_{[1]}(\pi_1(s)) = \Phi'^{-1}_{[1]}(\pi_1(s'))$ in each case. The case $i = 2$ is shown similarly.

(Var) We have $s = C[x \wedge y]$, $s' = C[z]\theta$ where $\theta = \{x := z, y := z\}$ is a substitution, and $\Phi' = \Phi \cup \{x \wedge y \mapsto z\}$ for some C, x, y. Then

$$\begin{array}{ll}
\Phi^{-1}_{[1]}(\pi_1(s)) & \\
= \Phi^{-1}_{[1]}(\pi_1(C[x \wedge y])) & \\
= \Phi^{-1}_{[1]}(\pi_1(C)[x]) & \text{by Lemma 7.6 (3)} \\
= (\Phi^{-1}_{[1]}(\pi_1(C)))[\Phi^{-1}_{[1]}(x),\ldots,\Phi^{-1}_{[1]}(x)] & \\
= (\Phi^{-1}_{[1]}(\pi_1(C\{y := z\})))[\ldots] & \text{by } y \in \mathscr{V}_2 \\
= (\Phi^{-1}_{[1]} \cup \{z \mapsto x\}(\pi_1(C\theta)))[\ldots] & \\
= (\Phi'^{-1}_{[1]}(\pi_1(C\theta)))[\Phi^{-1}_{[1]}(x),\ldots,\Phi^{-1}_{[1]}(x)] & \\
= (\Phi'^{-1}_{[1]}(\pi_1(C\theta)))[\Phi^{-1}_{[1]} \cup \{z \mapsto x\}(z),\ldots] & \\
= (\Phi'^{-1}_{[1]}(\pi_1(C\theta)))[\Phi'^{-1}_{[1]}(z),\ldots,\Phi'^{-1}_{[1]}(z)] & \\
= \Phi'^{-1}_{[1]}(\pi_1(C\theta[z])) & \\
= \Phi'^{-1}_{[1]}(\pi_1(C[z]\theta)) & \\
= \Phi'^{-1}_{[1]}(\pi_1(s')) &
\end{array}$$

Clearly, conditions (1),(2) hold for Φ' and s'.

(Div) We have $s = C[p(s_1,\ldots,s_n) \wedge p(t_1,\ldots,t_n)]$ and $s' = C[p(s_1 \wedge t_1,\ldots,s_n \wedge t_n)]$ for some

$C, p, s_1, \dots, t_n$ and $\Phi = \Phi'$. Then

$$\begin{aligned}
&\Phi^{-1}_{[1]}(\pi_1(s))\\
&= \Phi^{-1}_{[1]}(\pi_1(C[p(s_1, \dots, s_n)\\
&\qquad\qquad \wedge p(t_1, \dots, t_n)]))\\
&= \Phi^{-1}_{[1]}(\pi_1(C[p(s_1, \dots, s_n))]))\\
&\qquad\qquad \text{by Lemma 7.6 (3)}\\
&= \Phi^{-1}_{[1]}(\pi_1(C[p(s_1 \wedge t_1, \dots, s_n \wedge t_n)]))\\
&\text{by applying Lemma 7.6 (3) repeatedly}\\
&= \Phi^{-1}_{[1]}(\pi_1(s'))\\
&= \Phi'^{-1}_{[1]}(\pi_1(s'))
\end{aligned}$$

Clearly, conditions (1),(2) hold for Φ' and s'.

(Gen) We have $s = C[C_1\langle s_1, \dots, s_n\rangle \wedge C_2\langle t_1, \dots, t_n\rangle]$, $s' = C[p(\alpha_1, \dots, \alpha_n)]$, $\Phi' = \Phi \cup \{C_1 \wedge C_2 \mapsto p\}$ for some $C, C_1, C_2, p, s_1, \dots, t_n$. Then

$$\begin{aligned}
&\Phi^{-1}_{[1]}(\pi_1(s))\\
&= \Phi^{-1}_{[1]}(\pi_1(C[C_1\langle s_1, \dots, s_n\rangle\\
&\qquad\qquad \wedge C_2\langle t_1, \dots, t_n\rangle]))\\
&= \Phi^{-1}_{[1]}(\pi_1(C[C_1\langle s_1, \dots, s_n\rangle]))\\
&\qquad\qquad \text{by Lemma 7.6 (3)}\\
&= \pi_1(\Phi^{-1}_{[1]}(C[C_1\langle s_1, \dots, s_n\rangle]))\\
&\qquad\qquad \text{by Lemma 7.6 (2)}\\
&= \pi_1(\Phi^{-1}_{[1]}(C)[\Phi^{-1}_{[1]}(C_1\langle s_1, \dots, s_n\rangle),\\
&\qquad\qquad \dots \Phi^{-1}_{[1]}(C_1\langle s_1, \dots, s_n\rangle)])\\
&= \pi_1(\Phi^{-1}_{[1]}(C)[C_1\langle s_1, \dots, s_n\rangle\\
&\qquad\qquad \dots C_1\langle s_1, \dots, s_n\rangle])
\end{aligned}$$

since variables in $\mathrm{dom}(\Phi^{-1}_{[1]})$ are fresh. We now show that $\pi_1(C_1\langle \dots s_i \dots\rangle) = \pi_1(C_1\langle \dots \Phi'^{-1}_{[1]}(\alpha_i) \dots\rangle))$ holds for any i. We distinguish three cases.

(a) Case of $\Box_i \in \mathscr{H}(C_1) \cap \mathscr{H}(C_2)$. Then

$$\begin{aligned}
& \pi_1(C_1\langle \dots s_i \dots\rangle)\\
= \; & \pi_1(C_1\langle \dots s_i \wedge t_i \dots\rangle)\\
= \; & \pi_1(C_1\langle \dots \Phi'^{-1}_{[1]}(s_i \wedge t_i) \dots\rangle)\\
= \; & \pi_1(C_1\langle \dots \Phi'^{-1}_{[1]}(\alpha_i) \dots\rangle)
\end{aligned}$$

(b) Case of $\Box_i \in \mathscr{H}(C_1) \setminus \mathscr{H}(C_2)$.

$$\begin{aligned}
& \pi_1(C_1\langle \dots s_i \dots\rangle)\\
= \; & \pi_1(C_1\langle \dots \Phi'^{-1}_{[1]}(\Phi_{[1]}(s_i)) \dots\rangle)\\
& \qquad\qquad \text{by Lemma 7.8}\\
= \; & \pi_1(C_1\langle \dots \Phi'^{-1}_{[1]}(\alpha_i) \dots\rangle)
\end{aligned}$$

(c) Case of $\Box_i \in \mathscr{H}(C_2)\setminus\mathscr{H}(C_1)$. Then since $\Box_i \notin \mathscr{H}(C_1)$, by Lemma 7.13, $\pi_1(C_1\langle \dots s_i \dots\rangle) = \pi_1(C_1\langle \dots \Phi'^{-1}_{[1]}(\alpha_i) \dots\rangle)$.

Hence

$$\begin{aligned}
&\pi_1(\Phi_{[1]}^{-1}(C)[C_1\langle s_1,\ldots,s_n\rangle,\\
&\qquad\qquad \ldots C_1\langle s_1,\ldots,s_n\rangle])\\
&=\pi_1(\Phi'^{-1}_{[1]}(C)[C_1\langle \Phi'^{-1}_{[1]}(\alpha_1),\ldots,\rangle,\\
&\qquad\qquad \ldots C_1\langle \Phi'^{-1}_{[1]}(\alpha_1),\ldots,\rangle])\\
&=\pi_1(\Phi'^{-1}_{[1]}(C)[\Phi'^{-1}_{[1]}(p(\alpha_1,\ldots,\alpha_n)),\\
&\qquad\qquad \ldots \Phi'^{-1}_{[1]}(p(\alpha_1,\ldots,\alpha_n))])\\
&=\pi_1(\Phi'^{-1}_{[1]}(C[p(\alpha_1,\ldots,\alpha_n)]))\\
&=\Phi'^{-1}_{[1]}(\pi_1(C[p(\alpha_1,\ldots,\alpha_n)]))\\
&\qquad\qquad\qquad\qquad \text{by Lemma 7.6 (2)}\\
&=\Phi'^{-1}_{[1]}(\pi_1(s'))
\end{aligned}$$

Clearly, conditions (1),(2) hold for Φ' and s'.

□

Now we have the following soundness theorem of **2nd-Gen**.

Theorem 7.15. *Suppose $\langle s\wedge t,\emptyset\rangle \overset{*}{\rightsquigarrow} \langle u,\Phi\rangle$ and $\mathscr{V}(s)\cap\mathscr{V}(t)=\emptyset$. If u is $\wedge$-free then u is a generalization of s and t. Moreover, $\Phi_{[1]}^{-1}(u)=s$ and $\Phi_{[2]}^{-1}(u)=t$.*

Proof. By the assumption $\mathscr{V}(s)\cap\mathscr{V}(t)=\emptyset$, we can apply Lemma 7.14 repeatedly so to obtain $\Phi_{[1]}^{-1}(\pi_1(u))=s$ and $\Phi_{[2]}^{-1}(\pi_2(u))=t$. Since u is $\wedge$-free, $\pi_1(u)=\pi_2(u)=u$ by Lemma 7.6 (1). Thus $\Phi_{[1]}^{-1}(u)=s$ and $\Phi_{[2]}^{-1}(u)=t$. This means that u is a generalization of s and t. □

7.2 Generalization of TRSs

In this section, we give the TRS generalization procedure **TRS-Gen** based on the term generalization procedure **2nd-Gen** given in the previous section. We also present heuristics to drop solutions of generalization useless for constructing transformation patterns.

TRS-Gen generalizes two TRSs with an input memorizing function by generalizing each rewrite rule in sequence. A rewrite rule is treated as a term pattern whose root symbol is $\to$ in **TRS-Gen**. A memorizing function which is an input of **TRS-Gen** is used to keep consistent with the preceding generalizations.

Definition 7.16. Let $\mathcal{R}_1=\{l_1\to r_1,\ \ldots,\ l_n\to r_n\}$ and $\mathcal{R}_2=\{l'_1\to r'_1,\ldots,l'_n\to r'_n\}$ be TRS patterns over $\mathscr{F}$ and $\to$ a special binary function symbol such that $\to\notin\mathscr{F}$. The TRS generalization procedure **TRS-Gen** is given as follows:

Input: TRS patterns $\mathcal{R}_1$ and $\mathcal{R}_2$ and
  a memorizing function Φ.
begin
1. Rename local variables so that sets of
   local variables of each rewrite rule
   in $\mathcal{R}_1$ and $\mathcal{R}_2$ are mutually disjoint.
2. $\Phi_0=\Phi$
3. For($i=0$ to $i=n$)
   begin
     Compute $\tilde{l}_i\to\tilde{r}_i$ where
     $\langle\to(l_i\wedge l'_i, r_i\wedge r'_i),\Phi_{i-1}\rangle \overset{*}{\rightsquigarrow} \langle\to(\tilde{l}_i,\tilde{r}_i),\Phi_i\rangle$
     using **2nd-Gen**.

end

4. Output $\tilde{\mathcal{R}} = \{\tilde{l}_1 \to \tilde{r}_1, \ldots, \tilde{l}_n \to \tilde{r}_n\}$ and Φ_n.

end

The following is a corollary of Theorem 7.15.

Theorem 7.17. *Let $\tilde{\mathcal{R}}$ and $\tilde{\Phi}$ be outputs of **TRS-Gen** whose inputs are $\mathcal{R}_1$, $\mathcal{R}_2$ and Φ. $\tilde{\mathcal{R}}$ is a generalization of $\mathcal{R}_1$ and $\mathcal{R}_2$. More precisely, $\Phi^{-1}_{[1]}(\tilde{\mathcal{R}}) = \mathcal{R}_1$, $\Phi^{-1}_{[2]}(\tilde{\mathcal{R}}) = \mathcal{R}_2$ (up to renaming local variables) and $\Phi \subseteq \tilde{\Phi}$.*

We have implemented **2nd-Gen** and **TRS-Gen** using modules of program transformation system RAPT and performed experiments. It turned out that our algorithms tend to produce many solutions which are obviously useless to make transformation patterns. For example, the number of solutions of a generalization of $\mathsf{sum}(\mathsf{cons}(x, xs))$ and $\mathsf{cat}(\mathsf{cons}(y, ys))$ is over 1,000. Furthermore, it contains many solutions such as $\mathsf{p}(\mathsf{sum}(\mathsf{cons}(x, xs)), \mathsf{cat}(\mathsf{cons}(y, ys)))$ which are obviously useless for transformation patterns.

Even if many solutions of generalization are obtained, they have to be enriched into correct templates by adding appropriate hypotheses in order to use for program transformation. Since such enrichment is not always possible, it is preferred that obviously useless solutions are omitted beforehand. Below, we report several heuristics which work well in our experiment.

We first introduce two notions that are necessary for describing our heuristics. A notion of I-match is useful to reduce possibilities of application of **Gen**.

Definition 7.18. Let $C \in \mathrm{T}^{\square}_n(\mathscr{F} \cup \mathscr{X})$ be an indexed context, and $t \in \mathrm{T}(\mathscr{F} \cup \mathscr{X}, \mathscr{V})$ a term pattern. We say C *I-matches to* t if there exist term patterns $s_1, \ldots, s_n$ such that $C\langle s_1, \ldots s_n\rangle = t$.

We note that the notion of I-match is a variant of the first-order matching, which is decidable and has a unique solution up to renaming local variables.

Definition 7.19. (1) The set of *positions* of a term s is a set $\mathrm{Pos}(s)$ of sequences of integers, which is inductively defined as follows: (i) If $s = x \in \mathscr{V}$, then $\mathrm{Pos}(s) = \{\epsilon\}$ where ϵ represents empty sequence; (ii) If $s = q(s_1, ..., s_n)$, then $\mathrm{Pos}(s) = \{\epsilon\} \cup \bigcup_{i=1}^{n}\{ip \mid p \in \mathrm{Pos}(s_i)\}$. (2) Let s be a term pattern. A position p of s is *shallower than* a position q of s if $|p| \le |q|$. The position p is *the shallowest and leftmost* in t if (i) p is the shallowest in t; (ii) for any shallowest position q such that $q \neq p$, there exist p', i, j, q_1, and q_2 such that $p = p'iq_1$, $q = p'jq_2$ and $i < j$.

Our heuristics are as follows:

H1 **Gen** is applied only when neither **Var** nor **Div** can be applied.

H2 For a coupled term pattern s and memorizing function Φ, we chose the shallowest and leftmost $\wedge$-top subterm to apply **2nd-Gen**.

H3 When $\langle C[C_1\langle s_1, \ldots, s_n\rangle \wedge C_2\langle t_1, \ldots, t_n\rangle], \Phi\rangle \rightsquigarrow \langle C[p(\alpha_1, \ldots, \alpha_n)], \Phi'\rangle$ applying **Gen**, we restrict that the depth of each indexed context C_1 and C_2 is equal to or less than 1.

H4 For $\langle C[s \wedge t], \Phi\rangle$, we choose C_1 and C_2 to apply **Gen** if there exists $C_1 \wedge C_2 \mapsto p \in \Phi$ such that C_1 I-matches to s and C_2 I-matches to t.

H5 When **H4** cannot be applied to $\langle C[s \wedge t], \Phi\rangle$, we choose C_1 and C_2 to apply **Gen**, if there exist C_1, $s_1, \ldots, s_n$, C_2, $t_1, \ldots, t_n$, k, and $C'_1 \wedge C'_2 \mapsto p \in \Phi$ such that $s = C_1\langle s_1, \ldots, s_n\rangle$, $t = C_2\langle t_1, \ldots, t_n\rangle$, and $\square_k \in \mathscr{H}(C_1) \cap \mathscr{H}(C_2)$, and C'_1 and C'_2 I-match s_k and t_k, respectively.

H6 When **H4** and **H5** cannot apply to $\langle C[s \wedge t], \Phi\rangle$, we choose arbitrary indexed contexts satisfying **H3** to apply **Gen**.

Gen can be applied even when **Var** or **Div** can be done. One can obtain more concrete generalizations by giving higher priority to **Var** and **Div** than **Gen**. Here, we say a term pattern s is more concrete than a term pattern t if there exists a term homomorphism φ such that $\varphi(t) = s$. For example, let x, y be local variables. Without heuristics, **Var** and **Gen** can be applied to a pair $\langle x \wedge y, \emptyset\rangle$. If **Var** is applied then the pair $\langle z, \{x \wedge y \mapsto z\}\rangle$ is obtained. If **Gen** is applied then the pair $\langle p(x, y), \{\Box_1 \wedge \Box_2 \mapsto p\}\rangle$ is obtained. The former is more concrete than the latter.

By **H3**, the number of possibilities of application for **Gen** is reduced drastically. For example, there are 225 possibilities for applying **Gen** to $\langle +(\mathsf{s}(x), y) \wedge \mathsf{app}(\mathsf{cons}(z, zs), ws), \Phi\rangle$ without our heuristics while 81 possibilities for applying **Gen** with heuristic **H3** according to our experiment. In our experiments, heuristic **H3** seems to work well. However, there may exist transformations which the depth defined in **H3** should be increased.

Intuitively, **H4** and **H5** force to generalize common patterns by the same pattern variables. In our experiments, one can obtain more concrete generalizations with helps of **H4** and **H5**. For example, pars of generalizations of $\mathsf{f}(\mathsf{f}(x))$ and $\mathsf{g}(\mathsf{g}(y))$ are $\mathsf{p}(\mathsf{q}(v))$ and $\mathsf{p}(\mathsf{p}(v))$. The latter is more concrete than the former and produced using **H4** and **H5**.

Below we demonstrate one of the derivations following our heuristics (Fig. 7.3).

Step (a): We choose the shallowest and leftmost $\wedge$-top subterm $+(\mathsf{s}(x), y) \wedge \mathsf{app}(\mathsf{cons}(z, zs), ws)$ to apply **2nd-Gen** by **H2**. **Var**, **Div**, **H4** and **H5** cannot apply to this subterm. So, we choose $C_1 = +(\Box_1, \Box_2)$ and $C_2 = \mathsf{app}(\Box_1, \Box_2)$ to apply **Gen** to this subterm. As mentioned before, there are 81 possibilities of applying **Gen** to this subterm.

Step (b): The shallowest and leftmost $\wedge$-top subterm is $\mathsf{s}(+(x, y)) \wedge \mathsf{cons}(z, \mathsf{app}(zs, ws))$. Since $+(\Box_1, \Box_2)$ I-matches to $+(x, y)$ and $\mathsf{app}(\Box_1, \Box_2)$ I-matches to $\mathsf{app}(zs, ws)$, we choose $C_1 = \mathsf{s}(\Box_1)$ and $C_2 = \mathsf{cons}(\Box_2, \Box_1)$ to apply **Gen** to this subterm by **H5**.

Step (c): The shallowest and leftmost $\wedge$-top subterm is $\mathsf{s}(x) \wedge \mathsf{cons}(z, zs)$. Since $\mathsf{s}(\Box_1)$ I-matches to $\mathsf{s}(x)$ and $\mathsf{cons}(\Box_2, \Box_1)$ I-matches to $\mathsf{cons}(z, zs)$, we choose $C_1 = \mathsf{s}(\Box_1)$ and $C_2 = \mathsf{cons}(\Box_2, \Box_1)$ to apply **Gen** to this subterm by **H4**.

Step (d): We apply **H2** and **H4** as the step (c).

Step (e): The shallowest and leftmost $\wedge$-top subterm is $x \wedge zs$. We apply **Var** to this subterm by **H1**.

Steps (f), (g), and (h): We apply **Var** in the way similar to the step (e).

Example 7.20. *Let $\mathcal{R}_{sum}$ and $\mathcal{R}_{cat}$ be TRSs which appear in Chapter 3. The following TRS pattern $\tilde{\mathcal{P}}$ is one of outputs of our implementation with heuristics whose inputs are $\mathcal{R}_{sum}$, $\mathcal{R}_{cat}$ and $\emptyset$:*

$$\tilde{\mathcal{P}} \begin{cases} \mathsf{p}(\mathsf{r}) & \to \quad \mathsf{q} \\ \mathsf{p}(\mathsf{p2}(u, v)) & \to \quad \mathsf{p1}(u, \mathsf{p}(v)) \\ \mathsf{p1}(\mathsf{q}, v_1) & \to \quad v_1 \\ \mathsf{p1}(\mathsf{p3}(v_7, v_4), v_8) & \to \quad \mathsf{p3}(\mathsf{p1}(v_7, v_8), v_4) \end{cases}$$

The TRS pattern $\tilde{\mathcal{P}}$ above is a generalization of $\mathcal{R}_{sum}$ and $\mathcal{R}_{cat}$.

7.3 Generalization of transformations

In this section, we discuss how to construct transformation templates using our generalization algorithm.

A pair $\langle \mathcal{R}, \mathcal{R}' \rangle$ of TRSs is called a *TRS transformation*. We usually write the TRS transformation $\langle \mathcal{R}, \mathcal{R}' \rangle$ as $\mathcal{R} \Rightarrow \mathcal{R}'$. A transformation pattern $\mathcal{P} \Rightarrow \mathcal{P}'$ is a generalization of TRS transformations $\mathcal{R}_1 \Rightarrow \mathcal{R}'_1$ and $\mathcal{R}_2 \Rightarrow \mathcal{R}'_2$ if there exist term homomorphisms φ_1, φ_2 such that $\varphi_i(\mathcal{P}) = \mathcal{R}_i$ and $\varphi_i(\mathcal{P}') = \mathcal{R}'_i$ $(i = 1, 2)$ up to renaming local variables.

Definition 7.21. Let $\mathcal{R}_1 \Rightarrow \mathcal{R}'_1$ and $\mathcal{R}_2 \Rightarrow \mathcal{R}'_2$ be TRS transformations where $|\mathcal{R}_1| = |\mathcal{R}_2|$, $|\mathcal{R}'_1| = |\mathcal{R}'_2|$. Here, $|\mathcal{R}|$ denotes the number of rewrite rules appearing in $\mathcal{R}$. The procedure **Trans-Gen** is given as follows:

Input: $\mathcal{R}_1 \Rightarrow \mathcal{R}'_1$ and $\mathcal{R}_2 \Rightarrow \mathcal{R}'_2$
begin

1. Compute $\mathcal{P}$ and Φ by applying **TRS-Gen** to $\mathcal{R}_1$, $\mathcal{R}_2$ and $\emptyset$.
2. Compute $\mathcal{P}'$ and Φ' by applying **TRS-Gen** to $\mathcal{R}'_1$, $\mathcal{R}'_2$ and Φ.
3. Output $\mathcal{P} \Rightarrow \mathcal{P}'$.

end

The following is a corollary of Theorem 7.17.

Theorem 7.22. *Let $\mathcal{R}_1 \Rightarrow \mathcal{R}'_1$ and $\mathcal{R}_2 \Rightarrow \mathcal{R}'_2$ be TRS transformations, and $\mathcal{P} \Rightarrow \mathcal{P}'$ an output of **Trans-Gen** whose inputs are $\mathcal{R}_1 \Rightarrow \mathcal{R}'_1$ and $\mathcal{R}_2 \Rightarrow \mathcal{R}'_2$. Then $\mathcal{P} \Rightarrow \mathcal{P}'$ is a generalization of $\mathcal{R}_1 \Rightarrow \mathcal{R}'_1$ and $\mathcal{R}_2 \Rightarrow \mathcal{R}'_2$.*

Example 7.23. Applying **Trans-Gen** to $\mathcal{R}_{sum} \Rightarrow \mathcal{R}'_{sum}$ and $\mathcal{R}_{cat} \Rightarrow \mathcal{R}'_{cat}$ which appear in Section 1, the transformation pattern $\tilde{\mathcal{P}} \Rightarrow \tilde{\mathcal{P}}'$ is produced where

$$\tilde{\mathcal{P}}' \left\{ \begin{array}{ll} \mathsf{p}(v_{11}) & \to \mathsf{p4}(v_{11}, \mathsf{q}) \\ \mathsf{p4}(\mathsf{r}, v_{14}) & \to v_{14} \\ \mathsf{p4}(\mathsf{p2}(v_{23}, v_{21}), v_{22}) \to & \\ & \mathsf{p4}(v_{21}, \mathsf{p1}(v_{22}, v_{23})) \\ \mathsf{p1}(\mathsf{q}, v_{26}) & \to v_{26} \\ \mathsf{p1}(\mathsf{p3}(v_{32}, v_{29}), v_{33}) \to & \\ & \mathsf{p3}(\mathsf{p1}(v_{32}, v_{33}), v_{29}) \end{array} \right.$$

and $\tilde{\mathcal{P}}$ is the TRS pattern which appears in Example 7.20. We note that there exists little difference between $\mathcal{P} \Rightarrow \mathcal{P}'$ which appears in Section 1 and $\tilde{\mathcal{P}} \Rightarrow \tilde{\mathcal{P}}'$. But both of them is a generalization of $\mathcal{R}_{sum} \Rightarrow \mathcal{R}'_{sum}$ and $\mathcal{R}_{cat} \Rightarrow \mathcal{R}'_{cat}$.

To verify the correctness of transformations automatically, correct templates have to be constructed. One has to look for an appropriate hypothesis to construct a correct template from transformation patterns generated by **Trans-Gen**.

Example 7.24. *Let $\tilde{\mathcal{P}} \Rightarrow \tilde{\mathcal{P}}'$ be the transformation pattern appearing in Example 7.23 and $\tilde{\mathcal{H}}$ the following hypothesis.*

$$\tilde{\mathcal{H}} \quad \left\{ \begin{array}{lcl} \mathsf{p1}(\mathsf{q}, y) & \approx & \mathsf{p1}(y, \mathsf{q}) \\ \mathsf{p1}(x, \mathsf{p1}(y, z)) & \approx & \mathsf{p1}(\mathsf{p1}(x, y), z) \end{array} \right.$$

It can be shown that the template $\langle \tilde{\mathcal{P}}, \tilde{\mathcal{P}}', \tilde{\mathcal{H}} \rangle$ is correct.

Let us consider another example of generalization.

Example 7.25. *The following TRS transformations $\mathcal{R}_{onesadd} \Rightarrow \mathcal{R}'_{onesadd}$ and $\mathcal{R}_{lenapp} \Rightarrow \mathcal{R}'_{lenapp}$ represent the well-known program transformation called fusion transformation.*

$$\mathcal{R}_{onesadd}\left\{\begin{array}{ll}\mathsf{onesadd}(x,y) & \to \mathsf{ones}(+(x,y))\\ \mathsf{ones}(\mathsf{0}) & \to \mathsf{nil}\\ \mathsf{ones}(\mathsf{s}(x)) & \to \\ & \quad \mathsf{cons}(\mathsf{s}(\mathsf{0}),\mathsf{ones}(x))\\ +(\mathsf{0},x) & \to x\\ +(\mathsf{s}(x),y) & \to \mathsf{s}(+(x,y))\end{array}\right.$$

$$\mathcal{R}'_{onesadd}\left\{\begin{array}{ll}\mathsf{onesadd}(\mathsf{0},u) & \to \mathsf{ones}(u)\\ \mathsf{onesadd}(\mathsf{s}(v),w) & \to \\ & \quad \mathsf{cons}(\mathsf{s}(\mathsf{0}),\mathsf{onesadd}(v,w))\\ \mathsf{ones}(\mathsf{0}) & \to \mathsf{nil}\\ \mathsf{ones}(\mathsf{s}(v)) & \to \\ & \quad \mathsf{cons}(\mathsf{s}(\mathsf{0}),\mathsf{ones}(v))\\ +(\mathsf{0},u) & \to u\\ +(\mathsf{s}(v),w) & \to \mathsf{s}(+(v,w))\end{array}\right.$$

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

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

Applying ***Trans-Gen*** *to $\mathcal{R}_{onesadd} \Rightarrow \mathcal{R}'_{onesadd}$ and $\mathcal{R}_{lenapp} \Rightarrow \mathcal{R}'_{lenapp}$, the transformation pattern $\mathcal{P}_1 \Rightarrow \mathcal{P}'_1$ is obtained where*

$$\mathcal{P}_1\left\{\begin{array}{ll}\mathsf{p}(v,w) & \to \mathsf{q}(\mathsf{r}(v,w))\\ \mathsf{q}(\mathsf{p2}) & \to \mathsf{p1}\\ \mathsf{q}(\mathsf{p4}(v_3,v_1)) & \to \mathsf{p3}(\mathsf{s}(\mathsf{0}),\mathsf{q}(v_3))\\ \mathsf{r}(\mathsf{p2},v_6) & \to v_6\\ \mathsf{r}(\mathsf{p4}(v_{12},v_9),v_{13}) & \to \mathsf{p4}(\mathsf{r}(v_{12},v_{13}),v_9)\end{array}\right.$$

$$\mathcal{P}'_1\left\{\begin{array}{ll}\mathsf{p}(\mathsf{p2},v_{16}) & \to \mathsf{q}(v_{16})\\ \mathsf{p}(\mathsf{p4}(v_{22},v_{19}),v_{23}) & \to \\ & \quad \mathsf{p3}(\mathsf{s}(\mathsf{0}),\mathsf{p}(v_{22},v_{23}))\\ \mathsf{q}(\mathsf{p2}) & \to \mathsf{p1}\\ \mathsf{q}(\mathsf{p4}(v_{27},v_{25})) & \to \mathsf{p3}(\mathsf{s}(\mathsf{0}),\mathsf{q}(v_{27}))\\ \mathsf{r}(\mathsf{p2},v_{30}) & \to v_{30}\\ \mathsf{r}(\mathsf{p4}(v_{36},v_{33}),v_{37}) & \to \mathsf{p4}(\mathsf{r}(v_{36},v_{37}),v_{33})\end{array}\right.$$

Note that the transformation pattern which is obtained from $\mathcal{R}_{onesadd} \Rightarrow \mathcal{R}'_{onesadd}$ or $\mathcal{R}_{lenapp} \Rightarrow \mathcal{R}'_{lenapp}$ by replacing function symbols with fresh pattern variables cannot be used as transformation pattern for the other TRS.

Example 7.26. *The TRS $\mathcal{R}_{doubleadd}$ is transformed to $\mathcal{R}_{doubleadd'}$ by the transformation pattern $\mathcal{P}_1 \Rightarrow \mathcal{P}'_1$ where*

$$\mathcal{R}_{doubleadd} \begin{cases} \mathsf{doubleadd}(x, y) \to \\ \qquad\qquad \mathsf{double}(+(x, y)) \\ \mathsf{double}(0) \qquad \to 0 \\ \mathsf{double}(\mathsf{s}(x)) \quad \to \\ \qquad\qquad \mathsf{s}(\mathsf{s}(\mathsf{double}(x))) \\ +(0, x) \qquad \to x \\ +(\mathsf{s}(x), y) \qquad \to \mathsf{s}(+(x, y)) \end{cases}$$

$$\mathcal{R}'_{doubleadd} \begin{cases} \mathsf{doubleadd}(0, v_{16}) \to \\ \qquad\qquad \mathsf{double}(v_{16}) \\ \mathsf{doubleadd}(\mathsf{s}(v_{22}), v_{23}) \to \\ \quad \mathsf{s}(\mathsf{s}(\mathsf{doubleadd}(v_{22}, v_{23}))) \\ \mathsf{double}(0) \qquad \to 0 \\ \mathsf{double}(\mathsf{s}(v_{27})) \to \\ \qquad\qquad \mathsf{s}(\mathsf{s}(\mathsf{double}(v_{27}))) \\ +(0, v_{30}) \qquad \to v_{30} \\ +(\mathsf{s}(v_{36}), v_{37}) \to \\ \qquad\qquad \mathsf{s}(+(v_{36}, v_{37})) \end{cases}$$

Example 7.27. *The TRS $\mathcal{R}_{el}$ is transformed to $\mathcal{R}'_{el}$ by the transformation pattern $\mathcal{P}_1 \Rightarrow \mathcal{P}'_1$ where*

$$\mathcal{R}_{el} \begin{cases} \mathsf{evenlenapp}(x, y) & \to \mathsf{evenlen}(\mathsf{app}(x, y)) \\ \mathsf{evenlen}(\mathsf{nil}) & \to \mathsf{true} \\ \mathsf{evenlen}(\mathsf{cons}(x, y)) & \to \mathsf{not}(\mathsf{evenlen}(y)) \\ \mathsf{app}(\mathsf{nil}, x) & \to x \\ \mathsf{app}(\mathsf{cons}(x, y), z) & \to \mathsf{cons}(x, \mathsf{app}(y, z)) \\ \mathsf{not}(\mathsf{true}) & \to \mathsf{false} \\ \mathsf{not}(\mathsf{false}) & \to \mathsf{true} \end{cases}$$

$$\mathcal{R}'_{el} \begin{cases} \mathsf{evenlenapp}(\mathsf{nil}, v_{16}) \quad \to \\ \qquad\qquad \mathsf{evenlen}(v_{16}) \\ \mathsf{evenlenapp}(\mathsf{cons}(v_{19}, v_{22}), v_{23}) \to \\ \qquad \mathsf{not}(\mathsf{evenlenapp}(v_{22}, v_{23})) \\ \mathsf{evenlen}(\mathsf{nil}) \qquad \to \mathsf{true} \\ \mathsf{evenlen}(\mathsf{cons}(v_{25}, v_{27})) \to \\ \qquad\qquad \mathsf{not}(\mathsf{evenlen}(v_{27})) \\ \mathsf{app}(\mathsf{nil}, v_{30}) \qquad \to v_{30} \\ \mathsf{app}(\mathsf{cons}(v_{33}, v_{36}), v_{37}) \to \\ \qquad \mathsf{cons}(v_{33}, \mathsf{app}(v_{36}, v_{37})) \\ \mathsf{not}(\mathsf{true}) \qquad \to \mathsf{false} \\ \mathsf{not}(\mathsf{false}) \qquad \to \mathsf{true} \end{cases}$$

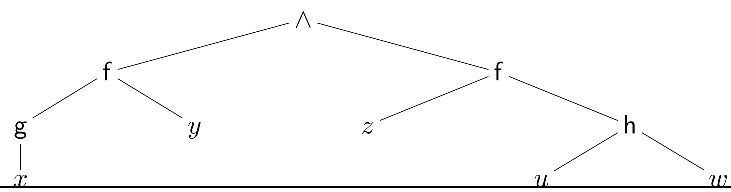
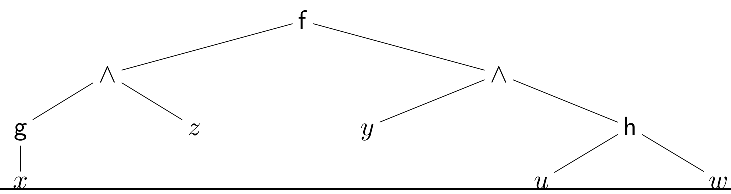
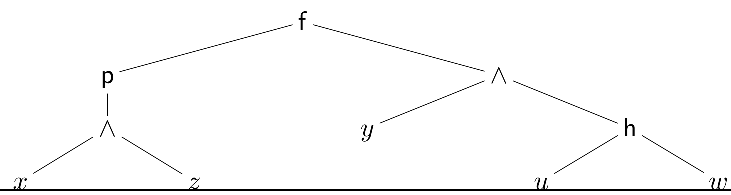
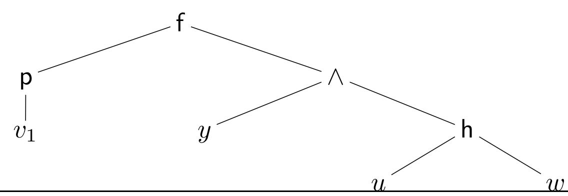
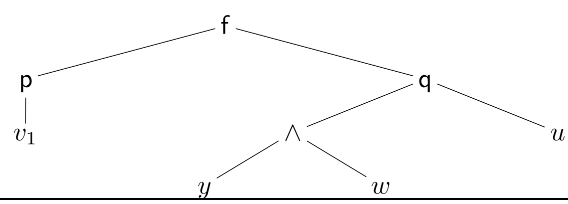
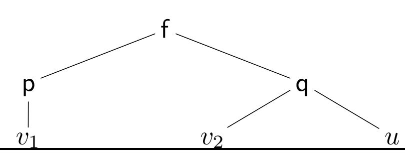
As mentioned before, templates have to be correct to verify the correctness of transformations automatically. In this example, it can be shown that the template $\langle \mathcal{P}_1, \mathcal{P}'_1, \emptyset \rangle$ is a correct template.

We now note about the implementation of our generalization algorithm. In our implementation, TRS transformations which are input of our algorithm are represented by pairs of two TRSs. The implementation of our generalization algorithm produces all solutions obtained under the heuristics **H1**$\sim$**H6**. Each output of our generalization algorithm is enumerated sequentially using the lazy evaluation technique.

7.4 Summary

In this chapter, we gave the term generalization algorithm **2nd-Gen** which generalizes two input terms. The soundness of **2nd-Gen** was shown in Theorem 7.15. We then extended **2nd-Gen** to the TRS generalization algorithm **TRS-Gen** which generalizes two input TRSs. Implementations of **2nd-Gen** and **TRS-Gen** showed that they produce huge number of solutions which includes many unexpected ones. We reported some heuristics (**H1**$\sim$**H6**) which reduce numbers of solutions. **Trans-Gen**, which generalizes two input transformations was given by extending **TRS-Gen**. We also gave examples of transformation templates produced by **Trans-Gen**. We checked through experiments that heuristics **H1**$\sim$**H6** help to construct correct templates.

Table 7.1: Example of generalization

step	coupled term	memorizing function
1		
2 (by II)		
3 (by III)		$\mathsf{g}(\Box_1) \wedge \Box_1 \mapsto \mathsf{p}$
4 (by I)		$\mathsf{g}(\Box_1) \wedge \Box_1 \mapsto \mathsf{p}$ $x \wedge z \mapsto v_1$
5 (by III)		$\mathsf{g}(\Box_1) \wedge \Box_1 \mapsto \mathsf{p}$ $x \wedge z \mapsto v_1$ $\Box_1 \wedge \mathsf{h}(\Box_2, \Box_1) \mapsto \mathsf{q}$
6 (by I)		$\mathsf{g}(\Box_1) \wedge \Box_1 \mapsto \mathsf{p}$ $x \wedge z \mapsto v_1$ $\Box_1 \wedge \mathsf{h}(\Box_2, \Box_1) \mapsto \mathsf{q}$ $y \wedge w \mapsto v_2$

$$\langle \rightarrow(+(\mathsf{s}(x),y) \wedge \mathsf{app}(\mathsf{cons}(z,zs),ws), \mathsf{s}(+(x,y))) \wedge \mathsf{cons}(z,\mathsf{app}(zs,ws)), \{\}\rangle$$

$$(a) \rightsquigarrow \langle \rightarrow(\mathsf{p}(\mathsf{s}(x) \wedge \mathsf{cons}(z,zs), y \wedge ws), \mathsf{s}(+(x,y)) \wedge \mathsf{cons}(z,\mathsf{app}(zs,ws))), \{+(\square_1,\square_2) \wedge \mathsf{app}(\square_1,\square_2) \mapsto \mathsf{p}\}\rangle$$

(by **H2**)

$$(b) \rightsquigarrow \langle \rightarrow(\mathsf{p}(\mathsf{s}(x) \wedge \mathsf{cons}(z,zs), y \wedge ws), \mathsf{q}(+(x,y) \wedge \mathsf{app}(zs,ws), z)), \left\{\begin{array}{ll} +(\square_1,\square_2) \wedge \mathsf{app}(\square_1,\square_2) & \mapsto \mathsf{p} \\ \mathsf{s}(\square_1) \wedge \mathsf{cons}(\square_2,\square_1) & \mapsto \mathsf{q} \end{array}\right\}\rangle$$

(by **H2** and **H5**)

$$(c) \rightsquigarrow \langle \rightarrow(\mathsf{p}(\mathsf{q}(x \wedge zs, z), y \wedge ws), \mathsf{q}(+(x,y) \wedge \mathsf{app}(zs,ws), z)), \left\{\begin{array}{ll} +(\square_1,\square_2) \wedge \mathsf{app}(\square_1,\square_2) & \mapsto \mathsf{p} \\ \mathsf{s}(\square_1) \wedge \mathsf{cons}(\square_2,\square_1) & \mapsto \mathsf{q} \end{array}\right\}\rangle$$

(by **H2** and **H4**)

$$(d) \rightsquigarrow \langle \rightarrow(\mathsf{p}(\mathsf{q}(x \wedge zs, z), y \wedge ws), \mathsf{q}(\mathsf{p}(x \wedge zs, y \wedge ws), z)), \left\{\begin{array}{ll} +(\square_1,\square_2) \wedge \mathsf{app}(\square_1,\square_2) & \mapsto \mathsf{p} \\ \mathsf{s}(\square_1) \wedge \mathsf{cons}(\square_2,\square_1) & \mapsto \mathsf{q} \end{array}\right\}\rangle$$

(by **H2** and **H4**)

$$(e) \rightsquigarrow \langle \rightarrow(\mathsf{p}(\mathsf{q}(u_1, z), y \wedge ws), \mathsf{q}(\mathsf{p}(x \wedge zs, y \wedge ws), z)), \left\{\begin{array}{ll} +(\square_1,\square_2) \wedge \mathsf{app}(\square_1,\square_2) & \mapsto \mathsf{p} \\ \mathsf{s}(\square_1) \wedge \mathsf{cons}(\square_2,\square_1) & \mapsto \mathsf{q} \\ x \wedge zs \mapsto u_1 & \end{array}\right\}\rangle$$

(by **H1** and **H2**)

$$(f) \rightsquigarrow \langle \rightarrow(\mathsf{p}(\mathsf{q}(u_1, z), u_2), \mathsf{q}(\mathsf{p}(x \wedge zs, y \wedge ws), z)), \left\{\begin{array}{ll} +(\square_1,\square_2) \wedge \mathsf{app}(\square_1,\square_2) & \mapsto \mathsf{p} \\ \mathsf{s}(\square_1) \wedge \mathsf{cons}(\square_2,\square_1) & \mapsto \mathsf{q} \\ x \wedge zs \mapsto u_1 \quad y \wedge ws \mapsto u_2 & \end{array}\right\}\rangle$$

(by **H1** and **H2**)

$$(g) \rightsquigarrow \langle \rightarrow(\mathsf{p}(\mathsf{q}(u_1, z), u_2), \mathsf{q}(\mathsf{p}(u_1, y \wedge ws), z)), \left\{\begin{array}{ll} +(\square_1,\square_2) \wedge \mathsf{app}(\square_1,\square_2) & \mapsto \mathsf{p} \\ \mathsf{s}(\square_1) \wedge \mathsf{cons}(\square_2,\square_1) & \mapsto \mathsf{q} \\ x \wedge zs \mapsto u_1 \quad y \wedge ws \mapsto u_2 & \end{array}\right\}\rangle$$

(by **H1** and **H2**)

$$(h) \rightsquigarrow \langle \rightarrow(\mathsf{p}(\mathsf{q}(u_1, z), u_2), \mathsf{q}(\mathsf{p}(u_1, u_2), z)), \left\{\begin{array}{ll} +(\square_1,\square_2) \wedge \mathsf{app}(\square_1,\square_2) & \mapsto \mathsf{p} \\ \mathsf{s}(\square_1) \wedge \mathsf{cons}(\square_2,\square_1) & \mapsto \mathsf{q} \\ x \wedge zs \mapsto u_1 \quad y \wedge ws \mapsto u_2 & \end{array}\right\}\rangle$$

(by **H1** and **H2**)

Figure 7.3: Example of **2nd-Gen** with heuristics

Chapter 8

Conclusion

In this thesis, we proposed a new framework of program transformation by templates based on term rewriting. Contributions of this thesis are listed as follows:

1. Introducing the notion of correct templates and giving sufficient conditions which guarantee the correctness of transformations by correct templates.

2. Proposing 2nd-order pattern matching algorithm **Match** and show its soundness and completeness.

3. Implementing our framework as RAPT and checking its operation through examples.

4. Proposing 2nd-order generalization algorithm **2nd-Gen** and showing its soundness.

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

We gave a sound and complete term pattern matching algorithm and showed that how our program transformation is automated using this algorithm. We now compare our framework for the program transformation and those based on lambda calculus [6, 8, 9, 11, 12, 22].

There is no significant difference between the second-order matching algorithm by Huet and Lang [12] 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[27]. By incorporating their ideas to our framework, more efficient and useful algorithm may be found.

For the correctness proof of the transformation, the 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.

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 correct 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 [8, 22]. The correctness of transformation in MAG system is based on Huet and Lang's framework [12]. 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 Sittampalam [7, 22]. 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 cooperated with automated theorem-proving techniques in the verification of hypotheses. RAPT involves an interesting integration of program-transformation and automated theorem-proving techniques.

We have proposed a 2nd-order generalization procedure **2nd-Gen** for term patterns and show its correctness. Based on this procedure, we have given a procedure to construct transformation templates from similar TRS transformations. By using some heuristics, we have constructed correct templates that are suitable for TRS transformations and correctness checking of transformations.

Plotkin proposed a first-order generalization algorithm[20]. The first-order generalization is simulated by treating local variables as fresh constant and permitting pattern variables instantiated only term patterns (i.e. indexed contexts without holes). Therefore, our framework is an extension of first-order generalization. To the best of our knowledge, there is no result of generalization which is specialized for program transformation.

The notion of program transformation by templates was originally introduced by Huet and Lang[12]. They showed the method to construct transformation templates manually. After their work, several results about program transformation by templates have been obtained[6, 8, 27]. In these works, no automated method to construct transformation templates has been proposed.

Bibliography

[1] F. Baader and T. Nipkow. *Term rewriting and all that*. Cambridge University Press, 1998.

[2] A. Bundy. The automation of proof by mathematical induction. In *Handbook of Automated Reasoning*, chapter 13, pages 845–911. Elsevier and MIT Press, 2001.

[3] R.M. Burstall and J. Darlington. A transformation system for developing recursive programs. *Journal of the ACM*, 24(1):44–67, 1977.

[4] H. Comon. Inductionless induction. In *Handbook of Automated Reasoning*, chapter 14, pages 913–962. Elsevier and MIT Press, 2001.

[5] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. *Tree automata techniques and applications*. 1997. `http://www.grappa.univ-lille3.fr/tata`.

[6] 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.

[7] 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.

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

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

[10] 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.

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

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

[13] D. Kapur, P. Narendran, and H. Zhang. On sufficient-completeness and related properties of term rewriting systems. *Acta Informatica*, 24(4):395–415, 1987.

[14] D. E. Knuth and P. B. Bendix. Simple word problems in universal algebra. In J. Leech, editor, *Computational problems in abstract algebra*, pages 263–297. Pergamon Press, 1970.

[15] A. Lazrek, P. Lescanne, and J. J. Thiel. Tools for proving inductive equalities, relative completeness, and ω-completeness. *Information and Computation*, 84:47–70, 1990.

[16] M. H. A. Newman. On theories with a combinatorial definition of 'equivalence'. *Annals of Mathematics*, 43(2):223–243, 1942.

[17] T. Nipkow and G. Weikum. A decidability result about sufficient-completeness of axiomatically specified abstract data types. In *Proceedings of the 6th GI-Conference on Theoretical Computer Science*, volume 145 of *LNCS*, pages 257–268. Springer-Verlag, 1983.

[18] R. Paige. Future directions in program transformations. *ACM Computing Surveys*, 28(4es):170, 1996.

[19] H. Partsch and R. Steinbrüggen. Program transformation systems. *ACM Computing Surveys*, 15(3):199–236, 1983.

[20] G. D. Plotkin. A note on inductive generalization. In *Machine Intelligence*, volume 5, chapter 8, pages 153–163. Edinbrgh University Press, 1969.

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

[22] G. Sittampalam. *Higher-order matching for program transformation*. PhD thesis, Magdalen College, 2001.

[23] Terese. *Term rewriting systems*. Cambridge University Press, 2003.

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

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

[26] P. Wadler. Deforestation: transforming programs to eliminate trees. *Theoretical Computer Science*, 73:231–248, 1990.

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

Publications

[i] Yuki Chiba, Takahito Aoto and Yoshihito Toyama,
Automatic Construction of Program Transformation Templates,
IPSJ Transactions on Programming, Vol.49, No.SIG 1 (PRO 35), pp.14–27, 2008.

[ii] Keiichirou Kusakari, and Yuki Chiba,
A Higher-Order Knuth-Bendix Procedure and its Applications,
IEICE Transactions on Information and Systems, Vol.E90–D, No.4, pp.707–715, Apr 2007.

[iii] Yuki Chiba, Takahito Aoto and Yoshihito Toyama,
Program Transformation by Templates: A Rewriting Framework,
IPSJ Transactions on Programming, Vol.47, No.SIG 16 (PRO 31), pp.52–65, 2006.

[iv] Yuki Chiba and Takahito Aoto,
RAPT: A Program Transformation System based on Term Rewriting,
In Proceedings of the 17th International Conference on Rewriting Techniques and Applications (RTA 2006), Seattle, WA, USA, Lecture Notes in Computer Science, Vol.4098, Springer-Verlag, pp.267–276, 2006.

[v] Yuki Chiba, Takahito Aoto and Yoshihito Toyama,
Introducing Sequence Variables in Program Transformation based on Templates,
In Proceedings of the Forum on Information Technology 2005 (FIT2005), Information Technology Letters, Vol.4, pp.5–8, 2005 (in japanese).

[vi] Yuki Chiba, Takahito Aoto and Yoshihito Toyama,
Program Transformation by Templates based on Term Rewriting,
In Proceedings of the 7th ACM-SIGPLAN International Symposium on Principles and Practice of Declarative Programming (PPDP 2005), ACP Press, pp.59–69, 2005.