Vol. 48 No. SIG 0(PRO 35) IPSJ Transactions on Programming 2007

Regular Paper

Automatic Construction of Program Transformation Templates

Yuki Chiba,[†] Takahito Aoto[†] and Yoshihito Toyama[†]

Program transformation by templates (Huet and Lang, 1978) is a technique to improve the efficiency of programs. In this technique, programs are transformed according to a given program transformation template. 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. Chiba et al. (2006) proposed a framework of program transformation by template based on term rewriting and automated verification of its correctness. Based on this framework, we 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.

1. Introduction

Automatic program transformation which intends to improve efficiency of input programs is one of the most fascinating techniques for programming languages[9),10)]. Several techniques for transforming functional programs have been developed[2),8),13)]. In particular, Huet and Lang[8)] introduced *program transformation by templates* based on lambda calculus, and several extensions of the technique has been proposed[6),7),14)].

Chiba et al. proposed a framework of program transformation by template based on *term rewriting*[3)~5)]. In their framework, program transformation is specified by some transformation pattern $\mathcal{P} \Rightarrow \mathcal{P}'$. A term rewriting system (*TRS* for short) is transformed according to a transformation pattern, by performing the pattern matching between the given TRS and the input part of the transformation pattern, and then by applying the result of the pattern matching to the output part of the transformation pattern (Fig. 1).

For example, the transformation pattern $\mathcal{P} \Rightarrow \mathcal{P}'$ represents a well-known program transformation from recursive programs to iterative programs where

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

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

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

$$\mathcal{R}_{sum} \left\{ \begin{array}{ll} \mathsf{sum}(\mathsf{nil}) & \to \mathsf{0} \\ \mathsf{sum}(\mathsf{cons}(x_1, ys_1)) & \to +(x_1, \mathsf{sum}(ys_1)) \\ +(\mathsf{0}, x_2) & \to x_2 \\ +(\mathsf{s}(x_3), y_3) & \to \mathsf{s}(+(x_3, y_3)) \end{array} \right.$$

The TRS pattern $\mathcal{P}$ matches to the TRS $\mathcal{R}_{sum}$ under the following *term homomorphism* φ, i.e., $\mathcal{R}_{sum} = \varphi(\mathcal{P})$.

$$\varphi = \left\{ \begin{array}{lll} \mathsf{f} \mapsto \mathsf{sum}(\Box_1), & u_1 \mapsto x_1, & u_6 \mapsto x_6, \\ \mathsf{g} \mapsto +(\Box_1, \Box_2), & v_1 \mapsto ys_1, & v_6 \mapsto y_6, \\ \mathsf{f}_1 \mapsto \mathsf{sum1}(\Box_1, \Box_2), & u_2 \mapsto x_2, & w_6 \mapsto z_6, \\ \mathsf{a} \mapsto \mathsf{nil}, & v_3 \mapsto x_3, & u_7 \mapsto x_7, \\ \mathsf{b} \mapsto \mathsf{0}, & w_3 \mapsto y_3, & v_8 \mapsto y_8, \\ \mathsf{c} \mapsto \mathsf{cons}(\Box_1, \Box_2), & u_4 \mapsto x_4, & w_8 \mapsto z_8 \\ \mathsf{d} \mapsto \mathsf{s}(\Box_2), & u_5 \mapsto x_5, & \end{array} \right\}$$

Thus, the TRS $\mathcal{R}_{sum}$ is transformed into the following TRS $\mathcal{R}'_{sum} = \varphi(\mathcal{P}')$.

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

Let us consider another example of program transformation. A program that computes the concatenation of a list of lists is specified by the

† Research Institute of Electrical Communication, Tohoku University

transformation pattern

$$\begin{tikzcd}
\mathcal{R} \arrow[d, Rightarrow, dashed] & & \mathcal{P} \arrow[ll, "\varphi", "matching"'] \arrow[d, Rightarrow] \\
\mathcal{R}' & & \mathcal{P}' \arrow[ll, dotted, "\varphi", "instantiation"']
\end{tikzcd}$$

Fig. 1 Overview of TRS transformation by transformation patterns

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

$$\mathcal{R}_{cat}\left\{\begin{array}{ll}
\mathsf{cat}(\mathsf{lnil}) & \to \mathsf{nil} \\
\mathsf{cat}(\mathsf{lcons}(x_1, ys_1)) & \to \\
 & \quad \mathsf{app}(x_1, \mathsf{cat}(ys_1)) \\
\mathsf{app}(\mathsf{nil}, x_2) & \to x_2 \\
\mathsf{app}(\mathsf{cons}(x_3, y_3), z_3) & \to \\
 & \quad \mathsf{cons}(x_3, \mathsf{app}(y_3, z_3))
\end{array}\right.$$

The TRS pattern $\mathcal{P}$ matches to the TRS $\mathcal{R}_{cat}$ under the following term homomorphism φ.

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

According to the template $\mathcal{P} \Rightarrow \mathcal{P}'$, $\mathcal{R}_{cat} = \varphi(\mathcal{P})$ can be transformed to the following TRS $\mathcal{R}'_{cat} = \varphi(\mathcal{P}')$.

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

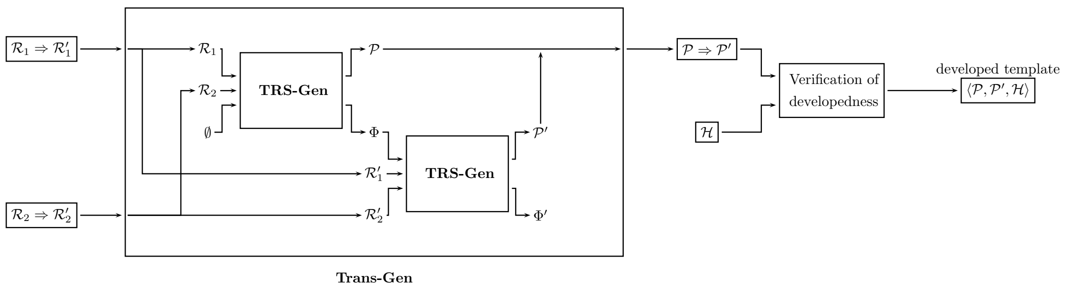
To apply the technique of program transformation by template, appropriate transformation patterns have to be constructed beforehand. Thus, it is important to introduce new transformation patterns in order to enhance the variety of program transformation. Up to our knowledge, however, few works discuss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

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. 2).

Our method is inspired by Plotkin's work[11)] 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. Chiba et al. introduced a method to prove the correctness of program transformation by template[3)~5)]. They 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 cor-



$\mathcal{R}_1 \Rightarrow \mathcal{R}'_1$
$\mathcal{R}_2 \Rightarrow \mathcal{R}'_2$
$\mathcal{R}_1$
$\mathcal{R}_2$
$\emptyset$
TRS-Gen
$\mathcal{P}$
Φ
$\mathcal{R}'_1$
$\mathcal{R}'_2$
TRS-Gen
$\mathcal{P}'$
Φ'
Trans-Gen
$\mathcal{P} \Rightarrow \mathcal{P}'$
$\mathcal{H}$
Verification of developedness
developed template
$\langle \mathcal{P}, \mathcal{P}', \mathcal{H} \rangle$

Fig. 2 Overview of the construction of a template

rectness of transformation.

For automatic verification of the correctness of transformations, Chiba et al. introduced a notion of *developed template* and gave sufficient conditions to verify the correctness of transformations by developed templates[3)~5)]. Developed templates are those that can be obtained using simple inference rules. Provided that the transformation template is developed, the correctness problem of a program transformation is reduced to a semi-decidable problem. For example, the template $\langle \mathcal{P}, \mathcal{P}', \mathcal{H} \rangle$ is developed where $\mathcal{P}$ and $\mathcal{P}'$ are those appeared before as the transformation pattern $\mathcal{P} \Rightarrow \mathcal{P}'$ and $\mathcal{H}$ is the following hypothesis:

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

Currently, no automatic method to produce developed 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 developedness to construct developed template (Fig. 2).

The rest of the paper is organized as follows. In Section 2, we recall basic notions in term rewriting and TRS transformation that will be used throughout this paper. In Section 3, we propose a non-deterministic 2nd-order generalization procedure of terms and prove its soundness. In Section 4, we give a TRS generalization procedure and report about an implementation of the procedure. We introduce several heuristics to omit obviously useless solutions and reduce the number of outputs of the generalization procedure. We then give a generalization procedure of templates and examples of construction of transformation templates in Section 5. We conclude our result in Section 6.

2. Preliminaries

This section introduces notions of term rewriting systems[1),12)] and program transformations by templates based on term rewriting[3)~5)].

Let $\mathscr{F}$, $\mathscr{X}$ and $\mathscr{V}$ be the sets of *function symbols*, *pattern variables* and *local variables*, respectively. Any function symbol and pattern variable $p \in \mathscr{F} \cup \mathscr{X}$ has its *arity* (denoted by $\mathrm{arity}(p)$). The set $\mathrm{T}(\mathscr{F} \cup \mathscr{X}, \mathscr{V})$ of *term patterns* (or just *patterns*) is defined by: (1) $\mathscr{V} \subseteq \mathrm{T}(\mathscr{F} \cup \mathscr{X}, \mathscr{V})$; and (2) $p(t_1, \ldots, t_n) \in \mathrm{T}(\mathscr{F} \cup \mathscr{X}, \mathscr{V})$ for any $p \in \mathscr{F} \cup \mathscr{X}$ such that $\mathrm{arity}(p) = n$ and $t_1, \ldots, t_n \in \mathrm{T}(\mathscr{F} \cup \mathscr{X}, \mathscr{V})$. For any term pattern s, the sets of function symbols, pattern variables and local variables in s are denoted by $\mathscr{F}(s)$, $\mathscr{X}(s)$ and $\mathscr{V}(s)$, respectively. For a term pattern $s = p(s_1, \ldots, s_n)$, the *root* symbol of s is p (denoted by $\mathrm{root}(s)$).

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

Special (indexed) constants $\Box_i$ $(i \geq 1)$ such that $\Box_i \notin \mathscr{F} \cup \mathscr{P} \cup \mathscr{V}$ are called (indexed) *holes*. The set of holes is denoted by $\mathscr{H}$. An *(indexed) context* C is an element of $\mathrm{T}(\mathscr{F} \cup \mathscr{X} \cup \mathscr{H}, \mathscr{V})$. $C[s_1, \ldots, s_n]$ is the result of C replacing $\Box_i$ with $s_1, \ldots, s_n$ from left to right. $C\langle s_1, \ldots, s_n \rangle$ is the result of C replacing $\Box_i$ by s_i for $i = 1, \ldots, n$ (*indexed replacement*). The set of indexed holes which appear in C is denoted by $\mathscr{H}(C)$. A context C with precisely one hole is denoted by $C[\,]$. The set of contexts is denoted by $\mathrm{T}^{\Box}(\mathscr{F} \cup \mathscr{X}, \mathscr{V})$; its subset $\mathrm{T}(\mathscr{F} \cup \mathscr{X} \cup \{\Box_i \mid 1 \leq$

$i \le n\}, \mathscr{V})$ is denoted by $\mathrm{T}^{\Box}_{n}(\mathscr{F} \cup \mathscr{X}, \mathscr{V})$. The sets of contexts $\mathrm{T}^{\Box}(\mathscr{F} \cup \mathscr{X})$ and $\mathrm{T}^{\Box}_{n}(\mathscr{F} \cup \mathscr{X})$ without local variables are defined accordingly.

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

A *transformation pattern* is a pair $\langle \mathcal{P}, \mathcal{P}'\rangle$ of two TRS patterns. We usually denote a transformation pattern $\langle \mathcal{P}, \mathcal{P}'\rangle$ as $\mathcal{P} \Rightarrow \mathcal{P}'$.

A mapping φ from $\mathscr{X} \cup \mathscr{V}$ to $T^{\Box}(\mathscr{F} \cup \mathscr{X}, \mathscr{V})$ is said to be a *term homomorphism* if (1) $\varphi(p) \in T^{\Box}_{\mathrm{arity}(p)}(\mathscr{F} \cup \mathscr{X})$ for any $p \in \mathrm{dom}_{\mathscr{X}}(\varphi)$, (2) $\varphi(x) \in \mathscr{V}$ for any $x \in \mathrm{dom}_{\mathscr{V}}(\varphi)$, and (3) φ is injective on $\mathrm{dom}_{\mathscr{V}}(\varphi)$, i.e., for any $x, y \in \mathrm{dom}_{\mathscr{V}}(\varphi)$, if $x \ne y$ then $\varphi(x) \ne \varphi(y)$, where $\mathrm{dom}_{\mathscr{X}}(\varphi) = \{p \in \mathscr{X} \mid \varphi(p) \ne p(\Box_1, \ldots, \Box_{\mathrm{arity}(p)})\}$ and $\mathrm{dom}_{\mathscr{V}}(\varphi) = \{x \in \mathscr{V} \mid \varphi(x) \ne x\}$. A term homomorphism φ is extended to a mapping $\hat{\varphi}$ over $\mathrm{T}(\mathscr{F} \cup \mathscr{X}, \mathscr{V})$ as follows:

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

We usually identify $\hat{\varphi}$ and φ. A term homomorphism is extended to a mapping on rewrite rules and equations in the obvious way.

A term pattern without pattern variables is called a *term*. The set of terms is denoted by $\mathrm{T}(\mathscr{F}, \mathscr{V})$. A TRS pattern over terms is called a *TRS*. Let $\mathcal{P} \Rightarrow \mathcal{P}'$ be a transformation pattern. We say a TRS $\mathcal{R}$ *is transformed into* $\mathcal{R}'$ *by* $\mathcal{P} \Rightarrow \mathcal{P}'$ if $\mathcal{R}$ and $\mathcal{R}'$ match $\mathcal{P}$ and $\mathcal{P}'$, respectively, by a term homomorphism φ, that is 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 $\mathcal{R}_{com}$. A pattern matching algorithm between $\mathcal{P}$ and $\mathcal{R}$ appears in 4).

3. 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 3.1 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 3.2 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 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**.

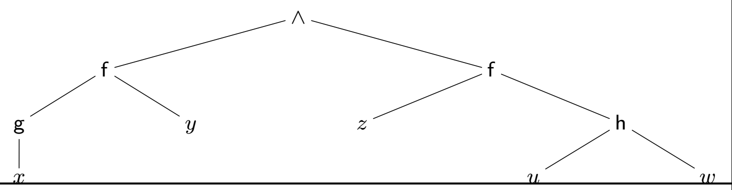
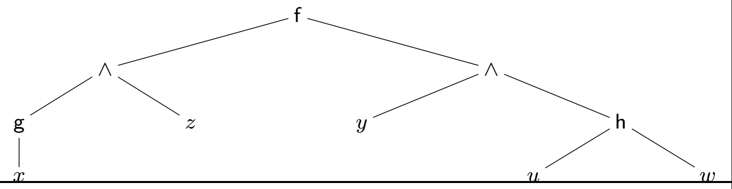
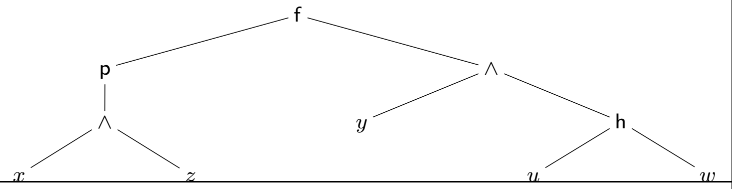
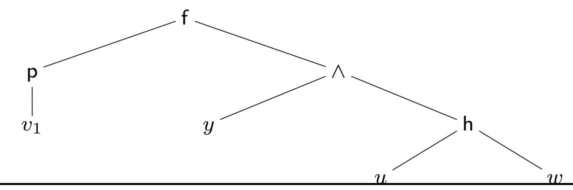
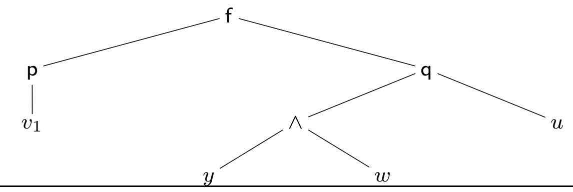
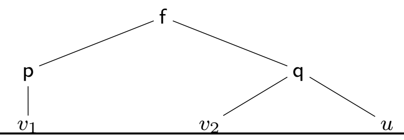
Initially, two input terms $\mathsf{f}(\mathsf{g}(x), y)$ and $\mathsf{f}(z, \mathsf{h}(u, w))$ are coupled into $\mathsf{f}(\mathsf{g}(x), y) \wedge \mathsf{f}(z, \mathsf{h}(u, w))$, using a special binary function symbol $\wedge$ (step 1). Since $\wedge$ indicates the position which will be generalized, 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.

Table 1 Example of generalization

step	coupled term	memorizing function
1		
2 (by II)		
3 (by III)		$\mathsf{g}(\square_1) \wedge \square_1 \mapsto \mathsf{p}$
4 (by I)		$\mathsf{g}(\square_1) \wedge \square_1 \mapsto \mathsf{p}$ $x \wedge z \mapsto v_1$
5 (by III)		$\mathsf{g}(\square_1) \wedge \square_1 \mapsto \mathsf{p}$ $x \wedge z \mapsto v_1$ $\square_1 \wedge \mathsf{h}(\square_2, \square_1) \mapsto \mathsf{q}$
6 (by I)		$\mathsf{g}(\square_1) \wedge \square_1 \mapsto \mathsf{p}$ $x \wedge z \mapsto v_1$ $\square_1 \wedge \mathsf{h}(\square_2, \square_1) \mapsto \mathsf{q}$ $y \wedge w \mapsto v_2$

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

Definition 3.3 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 3.4 Let t be a coupled term pattern. For $i = 1, 2$, the (first and second)

Var

$$\frac{C[x \wedge y],\ \Phi}{C[z]\theta,\ \Phi \cup \{x \wedge y \mapsto z\}}$$

if either
(1) $\Phi(x \wedge y) = z$ or
(2) $x \notin \mathrm{range}(\Phi^{-1}_{[1]})$, $y \notin \mathrm{range}(\Phi^{-1}_{[2]})$, and z is a fresh local variable
where $\theta = \{x := z, y := z\}$ is a substitution.

Div

$$\frac{C[p(s_1, \ldots, s_n) \wedge p(t_1, \ldots, t_n)],\ \Phi}{C[p(s_1 \wedge t_1, \ldots, s_n \wedge t_n)],\ \Phi}$$

if $p \in \mathscr{F} \cup \mathscr{X}$

Gen

$$\frac{C[C_1\langle s_1, \ldots, s_n\rangle \wedge C_2\langle t_1, \ldots, t_n\rangle],\ \Phi}{C[p(\alpha_1, \ldots, \alpha_n)],\ \Phi \cup \{C_1 \wedge C_2 \mapsto p\}}$$

if either $\Phi(C_1 \wedge C_2) = p$ or
(1) $C_1, C_2 \in T^{\square}_n(\mathscr{F} \cup \mathscr{X}), C_1 \neq C_2$,
(2) p is a fresh (n-ary) pattern variable
(3) $C_1 \wedge C_2 \notin \mathrm{dom}(\Phi)$
(4) $\mathscr{H}(C_1) \cup \mathscr{H}(C_2) = \{\square_1, \ldots, \square_n\}$, and
(5) $\alpha_i = \begin{cases} s_i \wedge t_i & \text{if } \square_i \in \mathscr{H}(C_1) \cap \mathscr{H}(C_2) \\ \Phi_{[1]}(s_i) & \text{if } \square_i \in \mathscr{H}(C_1) \setminus \mathscr{H}(C_2) \\ \Phi_{[2]}(t_i) & \text{if } \square_i \in \mathscr{H}(C_2) \setminus \mathscr{H}(C_1) \end{cases}$

Fig. 3 Inference rules of **2nd-Gen**

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 3.5 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 3.6 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 3.7 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 $C_1 \neq C_2$, $C_1, C_2 \in T^{\square}_n(\mathscr{F} \cup \mathscr{X})$, 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 $\Phi^{-1}_{[i]} = \{u \mapsto s_i \mid s_1 \wedge s_2 \mapsto u \in \Phi\}$, 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 3.8 Let Φ be a memorizing function. Let s be a $\wedge$-free term such that $\mathscr{V}(s) \cap \mathrm{range}(\Phi) = \emptyset$. Then $\Phi^{-1}_{[i]}(\Phi_{[i]}(s)) = s$.

The generalization procedure **2nd-Gen** works on pairs $\langle s, \Phi\rangle$ of a coupled term pattern s and a memorizing function Φ. Figure 3 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 3. 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 3.9 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 3.10 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 3.11 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 3.12 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 $w(p(\alpha_1, \ldots, \alpha_n)) = [|s_i| + |t_i| \mid \Box_i \in \mathscr{H}(C_1) \cap \mathscr{H}(C_2)]$ 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 $\Box_i \in \mathscr{H}(C_1) \cap \mathscr{H}(C_2)$ 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 $\Box_i \in \mathscr{H}(C_1) \cap \mathscr{H}(C_2)$. 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 $C_1 = \Box_1 = C_2$, thus this case does not happen by the condition of the inference rule.

□

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 3.13 For any indexed context C such that $\Box_i \notin \mathscr{H}(C)$ 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 3.14 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

$$\begin{aligned}
&\Phi^{-1}_{[1]}(\pi_1(s))\\
&= \Phi^{-1}_{[1]}(\pi_1(C[x \wedge y]))\\
&= \Phi^{-1}_{[1]}(\pi_1(C)[x]) \qquad \text{by Lemma 3.6 (3)}\\
&= (\Phi^{-1}_{[1]}(\pi_1(C)))[\Phi^{-1}_{[1]}(x), \ldots, \Phi^{-1}_{[1]}(x)]\\
&= (\Phi^{-1}_{[1]}(\pi_1(C\{y := z\})))[\ldots] \quad \text{by } y \in \mathscr{V}_2\\
&= (\Phi^{-1}_{[1]} \cup \{z \mapsto x\}(\pi_1(C\theta)))[\ldots]\\
&= (\Phi'^{-1}_{[1]}(\pi_1(C\theta)))[\Phi^{-1}_{[1]}(x), \ldots, \Phi^{-1}_{[1]}(x)]\\
&= (\Phi'^{-1}_{[1]}(\pi_1(C\theta)))[\Phi^{-1}_{[1]} \cup \{z \mapsto x\}(z), \ldots]\\
&= (\Phi'^{-1}_{[1]}(\pi_1(C\theta)))[\Phi'^{-1}_{[1]}(z), \ldots, \Phi'^{-1}_{[1]}(z)]\\
&= \Phi'^{-1}_{[1]}(\pi_1(C\theta[z]))\\
&= \Phi'^{-1}_{[1]}(\pi_1(C[z]\theta))\\
&= \Phi'^{-1}_{[1]}(\pi_1(s'))
\end{aligned}$$

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

$$\begin{aligned}
&\Phi^{-1}_{[1]}(\pi_1(s))\\
&= \Phi^{-1}_{[1]}(\pi_1(C[p(s_1, \ldots, s_n)\\
&\qquad\qquad \wedge p(t_1, \ldots, t_n)]))\\
&= \Phi^{-1}_{[1]}(\pi_1(C[p(s_1, \ldots, s_n))])) \\
&\qquad\qquad \text{by Lemma 3.6 (3)}\\
&= \Phi^{-1}_{[1]}(\pi_1(C[p(s_1 \wedge t_1, \ldots, s_n \wedge t_n)]))\\
&\text{by applying Lemma 3.6 (3) repeatedly}\\
&= \Phi^{-1}_{[1]}(\pi_1(s'))\\
&= \Phi'^{-1}_{[1]}(\pi_1(s'))
\end{aligned}$$

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

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

$$\begin{aligned}
&\Phi^{-1}_{[1]}(\pi_1(s))\\
&= \Phi^{-1}_{[1]}(\pi_1(C[C_1\langle s_1, \ldots, s_n\rangle\\
&\qquad\qquad \wedge C_2\langle t_1, \ldots, t_n\rangle]))\\
&= \Phi^{-1}_{[1]}(\pi_1(C[C_1\langle s_1, \ldots, s_n\rangle]))\\
&\qquad\qquad \text{by Lemma 3.6 (3)}\\
&= \pi_1(\Phi^{-1}_{[1]}(C[C_1\langle s_1, \ldots, s_n\rangle]))\\
&\qquad\qquad \text{by Lemma 3.6 (2)}\\
&= \pi_1(\Phi^{-1}_{[1]}(C)[\Phi^{-1}_{[1]}(C_1\langle s_1, \ldots, s_n\rangle),\\
&\qquad\qquad \ldots \Phi^{-1}_{[1]}(C_1\langle s_1, \ldots, s_n\rangle)])\\
&= \pi_1(\Phi^{-1}_{[1]}(C)[C_1\langle s_1, \ldots, s_n\rangle\\
&\qquad\qquad \ldots C_1\langle s_1, \ldots, s_n\rangle])
\end{aligned}$$

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

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

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

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

$$\begin{aligned}
&\pi_1(C_1\langle \ldots s_i \ldots\rangle)\\
&= \pi_1(C_1\langle \ldots \Phi'^{-1}_{[1]}(\Phi_{[1]}(s_i)) \ldots\rangle)\\
&\qquad\qquad \text{by Lemma 3.8}\\
&= \pi_1(C_1\langle \ldots \Phi'^{-1}_{[1]}(\alpha_i) \ldots\rangle)
\end{aligned}$$

(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 3.13, $\pi_1(C_1\langle \ldots s_i \ldots\rangle) = \pi_1(C_1\langle \ldots \Phi'^{-1}_{[1]}(\alpha_i) \ldots\rangle)$.

Hence

$$\begin{aligned}
&\pi_1(\Phi^{-1}_{[1]}(C)[C_1\langle s_1, \ldots, s_n\rangle,\\
&\qquad\qquad \ldots C_1\langle s_1, \ldots, s_n\rangle])\\
&= \pi_1(\Phi'^{-1}_{[1]}(C)[C_1\langle \Phi'^{-1}_{[1]}(\alpha_1), \ldots, \rangle,\\
&\qquad\qquad \ldots C_1\langle \Phi'^{-1}_{[1]}(\alpha_1), \ldots, \rangle])\\
&= \pi_1(\Phi'^{-1}_{[1]}(C)[\Phi'^{-1}_{[1]}(p(\alpha_1, \ldots, \alpha_n)),\\
&\qquad\qquad \ldots \Phi'^{-1}_{[1]}(p(\alpha_1, \ldots, \alpha_n))])\\
&= \pi_1(\Phi'^{-1}_{[1]}(C[p(\alpha_1, \ldots, \alpha_n)]))\\
&= \Phi'^{-1}_{[1]}(\pi_1(C[p(\alpha_1, \ldots, \alpha_n)]))\\
&\qquad\qquad \text{by Lemma 3.6 (2)}\\
&= \Phi'^{-1}_{[1]}(\pi_1(s'))
\end{aligned}$$

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

$\Box$

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

Theorem 3.15 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 $\Phi^{-1}_{[2]}(u) = t$.

Proof. By the assumption $\mathscr{V}(s) \cap \mathscr{V}(t) = \emptyset$, we can apply Lemma 3.14 repeatedly so to obtain $\Phi^{-1}_{[1]}(\pi_1(u)) = s$ and $\Phi^{-1}_{[2]}(\pi_2(u)) = t$. Since u is $\wedge$-free, $\pi_1(u) = \pi_2(u) = u$ by Lemma 3.6 (1). Thus $\Phi^{-1}_{[1]}(u) = s$ and $\Phi^{-1}_{[2]}(u) = t$. This means that u is a generalization of s and t. $\square$

4. 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 4.1 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 $\Phi$.
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 $\Phi_n$.
end
```

The following is a corollary of Theorem 3.15.

Theorem 4.2 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[3)~5)] 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 developed 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 4.3 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 4.4 (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 $|p| \leq |q|$. 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 $\Box_k \in \mathscr{H}(C_1) \cap \mathscr{H}(C_2)$, 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. 4).

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 4.5 Let $\mathcal{R}_{sum}$ and $\mathcal{R}_{cat}$ be TRSs which appear in Section 1. 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 $\emptyset$:

$$\tilde{\mathcal{P}} \left\{ \begin{array}{lcl} \mathsf{p}(\mathsf{r}) & \to & \mathsf{q} \\ \mathsf{p}(\mathsf{p2}(u, v)) & \to & \mathsf{p1}(u, \mathsf{p}(v)) \\ \mathsf{p1}(\mathsf{q}, v_1) & \to & v_1 \\ \mathsf{p1}(\mathsf{p3}(v_7, v_4), v_8) & \to & \mathsf{p3}(\mathsf{p1}(v_7, v_8), v_4) \end{array} \right.$$

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

5. 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 5.1 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 $\mathcal{R}_2 \Rightarrow \mathcal{R}'_2$

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

$$(a) \rightsquigarrow \langle \to(\mathsf{p}(\mathsf{s}(x) \wedge \mathsf{cons}(z, zs), y \wedge ws), \mathsf{s}(+(x, y)) \wedge \mathsf{cons}(z, \mathsf{app}(zs, ws))), \{+(\Box_1, \Box_2) \wedge \mathsf{app}(\Box_1, \Box_2) \mapsto \mathsf{p}\}\rangle \qquad \text{(by } \mathbf{H2}\text{)}$$

$$(b) \rightsquigarrow \langle \to(\mathsf{p}(\mathsf{s}(x) \wedge \mathsf{cons}(z, zs), y \wedge ws), \mathsf{q}(+(x, y) \wedge \mathsf{app}(zs, ws), z)), \left\{ \begin{array}{ll} +(\Box_1, \Box_2) \wedge \mathsf{app}(\Box_1, \Box_2) & \mapsto \mathsf{p} \\ \mathsf{s}(\Box_1) \wedge \mathsf{cons}(\Box_2, \Box_1) & \mapsto \mathsf{q} \end{array} \right\}\rangle \qquad \text{(by } \mathbf{H2} \text{ and } \mathbf{H5}\text{)}$$

$$(c) \rightsquigarrow \langle \to(\mathsf{p}(\mathsf{q}(x \wedge zs, z), y \wedge ws), \mathsf{q}(+(x, y) \wedge \mathsf{app}(zs, ws), z)), \left\{ \begin{array}{ll} +(\Box_1, \Box_2) \wedge \mathsf{app}(\Box_1, \Box_2) & \mapsto \mathsf{p} \\ \mathsf{s}(\Box_1) \wedge \mathsf{cons}(\Box_2, \Box_1) & \mapsto \mathsf{q} \end{array} \right\}\rangle \qquad \text{(by } \mathbf{H2} \text{ and } \mathbf{H4}\text{)}$$

$$(d) \rightsquigarrow \langle \to(\mathsf{p}(\mathsf{q}(x \wedge zs, z), y \wedge ws), \mathsf{q}(\mathsf{p}(x \wedge zs, y \wedge ws), z)), \left\{ \begin{array}{ll} +(\Box_1, \Box_2) \wedge \mathsf{app}(\Box_1, \Box_2) & \mapsto \mathsf{p} \\ \mathsf{s}(\Box_1) \wedge \mathsf{cons}(\Box_2, \Box_1) & \mapsto \mathsf{q} \end{array} \right\}\rangle \qquad \text{(by } \mathbf{H2} \text{ and } \mathbf{H4}\text{)}$$

$$(e) \rightsquigarrow \langle \to(\mathsf{p}(\mathsf{q}(u_1, z), y \wedge ws), \mathsf{q}(\mathsf{p}(x \wedge zs, y \wedge ws), z)), \left\{ \begin{array}{ll} +(\Box_1, \Box_2) \wedge \mathsf{app}(\Box_1, \Box_2) & \mapsto \mathsf{p} \\ \mathsf{s}(\Box_1) \wedge \mathsf{cons}(\Box_2, \Box_1) & \mapsto \mathsf{q} \\ x \wedge zs \mapsto u_1 & \end{array} \right\}\rangle \qquad \text{(by } \mathbf{H1} \text{ and } \mathbf{H2}\text{)}$$

$$(f) \rightsquigarrow \langle \to(\mathsf{p}(\mathsf{q}(u_1, z), u_2), \mathsf{q}(\mathsf{p}(x \wedge zs, y \wedge ws), z)), \left\{ \begin{array}{ll} +(\Box_1, \Box_2) \wedge \mathsf{app}(\Box_1, \Box_2) & \mapsto \mathsf{p} \\ \mathsf{s}(\Box_1) \wedge \mathsf{cons}(\Box_2, \Box_1) & \mapsto \mathsf{q} \\ x \wedge zs \mapsto u_1 \quad y \wedge ws \mapsto u_2 & \end{array} \right\}\rangle \qquad \text{(by } \mathbf{H1} \text{ and } \mathbf{H2}\text{)}$$

$$(g) \rightsquigarrow \langle \to(\mathsf{p}(\mathsf{q}(u_1, z), u_2), \mathsf{q}(\mathsf{p}(u_1, y \wedge ws), z)), \left\{ \begin{array}{ll} +(\Box_1, \Box_2) \wedge \mathsf{app}(\Box_1, \Box_2) & \mapsto \mathsf{p} \\ \mathsf{s}(\Box_1) \wedge \mathsf{cons}(\Box_2, \Box_1) & \mapsto \mathsf{q} \\ x \wedge zs \mapsto u_1 \quad y \wedge ws \mapsto u_2 & \end{array} \right\}\rangle \qquad \text{(by } \mathbf{H1} \text{ and } \mathbf{H2}\text{)}$$

$$(h) \rightsquigarrow \langle \to(\mathsf{p}(\mathsf{q}(u_1, z), u_2), \mathsf{q}(\mathsf{p}(u_1, u_2), z)), \left\{ \begin{array}{ll} +(\Box_1, \Box_2) \wedge \mathsf{app}(\Box_1, \Box_2) & \mapsto \mathsf{p} \\ \mathsf{s}(\Box_1) \wedge \mathsf{cons}(\Box_2, \Box_1) & \mapsto \mathsf{q} \\ x \wedge zs \mapsto u_1 \quad y \wedge ws \mapsto u_2 & \end{array} \right\}\rangle \qquad \text{(by } \mathbf{H1} \text{ and } \mathbf{H2}\text{)}$$

Fig. 4 Example of **2nd-Gen** with heuristics

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

The following is a corollary of Theorem 4.2.

Theorem 5.2 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 5.3 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

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

and $\tilde{\mathcal{P}}$ is the TRS pattern which appears in Example 4.5. 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, developed templates have to be constructed[3)~5)]. One has to look for an appropriate hypothesis to construct a developed template from transformation patterns generated by **Trans-Gen**.

Example 5.4 Let $\tilde{\mathcal{P}} \Rightarrow \tilde{\mathcal{P}}'$ be the transformation pattern appearing in Example 5.3 and $\tilde{\mathcal{H}}$ the following hypothesis.

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

It can be shown that the template $\langle \tilde{\mathcal{P}}, \tilde{\mathcal{P}}', \tilde{\mathcal{H}} \rangle$ is developed[4),5)].

Let us consider another example of generalization.

Example 5.5 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.

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

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

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

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

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

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

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

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

$$\mathcal{R}_{doubleadd} \left\{ \begin{array}{ll} \mathsf{doubleadd}(x, y) \to & \\ \qquad \mathsf{double}(+(x, y)) & \\ \mathsf{double}(0) & \to 0 \\ \mathsf{double}(\mathsf{s}(x)) & \to \\ & \quad \mathsf{s}(\mathsf{s}(\mathsf{double}(x))) \\ +(0, x) & \to x \\ +(\mathsf{s}(x), y) & \to \mathsf{s}(+(x, y)) \end{array} \right.$$

$$\mathcal{R}'_{doubleadd}\left\{\begin{array}{ll}\mathsf{doubleadd}(\mathsf{0}, v_{16}) \to & \\ \quad \mathsf{double}(v_{16}) & \\ \mathsf{doubleadd}(\mathsf{s}(v_{22}), v_{23}) \to & \\ \quad \mathsf{s}(\mathsf{s}(\mathsf{doubleadd}(v_{22}, v_{23}))) & \\ \mathsf{double}(\mathsf{0}) & \to \mathsf{0} \\ \mathsf{double}(\mathsf{s}(v_{27})) \to & \\ \quad \mathsf{s}(\mathsf{s}(\mathsf{double}(v_{27}))) & \\ +(\mathsf{0}, v_{30}) & \to v_{30} \\ +(\mathsf{s}(v_{36}), v_{37}) \to & \\ \quad \mathsf{s}(+(v_{36}, v_{37})) & \end{array}\right.$$

Example 5.7 The TRS $\mathcal{R}_{el}$ is transformed to $\mathcal{R}'_{el}$ by the transformation pattern $\mathcal{P}_1 \Rightarrow \mathcal{P}'_1$ where

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

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

As mentioned before, templates have to be developed 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 developed template[4),5)].

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**∼**H6**. Each output of our generalization algorithm is enumerated sequentially using the lazy evaluation technique.

6. Conclusion

We have proposed the 2nd-order generalization procedure **2nd-Gen** for term patterns and show its soundness. Based on this procedure, we have given a procedure to construct transformation patterns from similar TRS transformations. By using some heuristics, the number of outputs of the generalization procedure is reduced and useless solutions are omitted. By adding appropriate hypotheses, we have also demonstrated that developed templates are obtained from transformation patterns produced using **Trans-Gen**.

Plotkin proposed a first-order generalization algorithm[11)]. 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[8)]. They showed the method to construct transformation templates manually. After their work, several results about program transformation by templates have been obtained[6),7),14)]. In these works, no automated method to construct transformation templates has been proposed.

Although soundness of the generalization procedure **2nd-Gen** was proved, proving completeness of **2nd-Gen** remains as a future work. In our framework, transformation templates are constructed manually from transformation patterns obtained by generalization procedure. We consider that it is interesting to attack the problem of constructing developed templates directly and automatically.

Acknowledgments

The authors would like to acknowledge their grateful thanks to an anonymous referee for useful comments and advices. The presentation of the paper has been improved very much by the constructive suggestions of the referee. This work was partially supported by grants from JSPS, Nos. 17700002 and 19500003.

References

1) Baader, F. and Nipkow, T.: *Term rewriting and all that*, Cambridge University Press (1998).
2) Burstall, R. and Darlington, J.: A transformation system for developing recursive programs, *Journal of the ACM*, Vol. 24, No. 1, pp. 44–67 (1977).

3) Chiba, Y. and Aoto, T.: RAPT: A program transformation system based on term rewriting, *Proceedings of the 17th International Conference on Rewriting Techniques and Applications*, LNCS, Vol. 4098, Springer-Verlag, pp. 267–276 (2006).

4) Chiba, Y., Aoto, T. and Toyama, Y.: Program transformation by templates based on term rewriting, *Proceedings of the 7th ACM-SIGPLAN International Conference on Principles and Practice of Declarative Programming (PPDP 2005)*, ACM Press, pp. 59–69 (2005).

5) Chiba, Y., Aoto, T. and Toyama, Y.: Program transformation by templates: A rewriting framework, *IPSJ Transactions on Programming*, Vol. 47, No. SIG 16 (PRO 31), pp. 52–65 (2006).

6) Curien, R., Qian, Z. and Shi, H.: Efficient second-order matching, *Proceedings of the 7th International Conference on Rewriting Techniques and Applications*, LNCS, Vol. 1103, Springer-Verlag, pp. 317–331 (1996).

7) de Moor, O. and Sittampalam, G.: Higher-order matching for program transformation, *Theoretical Computer Science*, Vol. 269, pp. 135–162 (2001).

8) Huet, G. and Lang, B.: Proving and applying program transformations expressed with second order patterns, *Acta Informatica*, Vol. 11, pp. 31–55 (1978).

9) Paige, R.: Future directions in program transformations, *ACM Computing Surveys*, Vol. 28, No. 4es, p. 170 (1996).

10) Partsch, H. and Steinbrüggen, R.: Program transformation systems., *ACM Computing Surveys*, Vol. 15, No. 3, pp. 199–236 (1983).

11) Plotkin, G. D.: A note on inductive generalization, *Machine Intelligence*, Vol. 5, Edinbrgh University Press, chapter 8, pp. 153–163 (1969).

12) Terese: *Term rewriting systems*, Cambridge University Press (2003).

13) Wadler, P.: Deforestation: transforming programs to eliminate trees, *Theoretical Computer Science*, Vol. 73, pp. 231–248 (1990).

14) Yokoyama, T., Hu, Z. and Takeichi, M.: Deterministic second-order patterns, *Information Processing Letters*, Vol. 89, No. 6, pp. 309–314 (2004).

(Received May 07, 19)
(Accepted August 01, 19)

Yuki China received his M.S. from Tohoku University in 2005. He is currently a doctoral student at the same university. His research interests include term rewriting and program transformation. He is a member of IPSJ and JSSST.

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

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