Transformation by Templates for Simply-Typed Term Rewriting

Yuki Chiba
School of Information Science, Japan Advanced Institute of Science and Technology
1-1 Asahidai, Nomi, Ishikawa, 923-1292, Japan
chiba@jaist.ac.jp

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

We extend a framework of program transformation by templates based on first order term rewriting (Chiba et al., 2005) to simply typed term rewriting (Yamada, 2001), which is a framework of higher order term rewriting. A pattern matching algorithm to apply templates for transforming a simply typed term rewriting system is given and the correctness of the algorithm is shown.

1 Program transformation by templates and simply typed term pattern

Huet and Lang [4] introduced a framework of program transformation by templates. In this framework, the second-order matching algorithm plays an important role—how to apply the transformation template to a given program is specified by the solution of the matching algorithm. Curien et al. [3] gave an efficient matching algorithm. Yokoyama et al. [7] presented a sufficient condition of patterns to have at most one solution. De Moor and Sittampalam [5] gave a matching algorithm containing third-order matching. In all of these frameworks, programs and program schemas are represented by lambda terms and higher-order substitutions are performed by β-reduction.

In Chiba et al. [1, 2], we introduced a framework of program transformation by templates *based on term rewriting*. Contrast to the framework mentioned above, programs and program schemas are given by term rewriting systems (TRSs for short) and TRS patterns, where TRS patterns is a TRS in which pattern variables are used in the place of function symbols. For example, a program transformation template $\langle \mathscr{P}, \mathscr{P}', \mathscr{H} \rangle$ is given like this:

$$\mathscr{P} = \left\{ \begin{array}{ll} \mathsf{f}(\mathsf{a}) & \to \mathsf{b} \\ \mathsf{f}(\mathsf{c}(u_1, v_1)) & \to \mathsf{g}(u_1, \mathsf{f}(v_1)) \\ \mathsf{g}(\mathsf{b}, u_2) & \to u_2 \\ \mathsf{g}(\mathsf{d}(u_3, v_3), w_3) & \to \mathsf{d}(u_3, \mathsf{g}(v_3, w_3)) \end{array} \right\}, \mathscr{P}' = \left\{ \begin{array}{ll} \mathsf{f}(u_4) & \to \mathsf{f}_1(u_4, \mathsf{b}) \\ \mathsf{f}_1(\mathsf{a}, u_5) & \to u_5 \\ \mathsf{f}_1(\mathsf{c}(u_6, v_6), w_6) & \to \mathsf{f}_1(v_6, \mathsf{g}(w_6, u_6)) \\ \mathsf{g}(\mathsf{b}, u_7) & \to u_7 \\ \mathsf{g}(\mathsf{d}(u_8, v_8), w_8) & \to \mathsf{d}(u_8, \mathsf{g}(v_8, w_8)) \end{array} \right\},$$

$$\mathscr{H} = \{ \quad \mathsf{g}(\mathsf{b}, u_1) \approx \mathsf{g}(u_1, \mathsf{b}), \quad \mathsf{g}(\mathsf{g}(u_2, v_2), w_2) \approx \mathsf{g}(u_2, \mathsf{g}(v_2, w_2)) \quad \}.$$

The TRS pattern $\mathscr{P}$ is a schema of input programs to be transformed and $\mathscr{P}'$ is a schema whose instantiations become the output programs. This template is used for a program transformation from recursive programs to iterative programs. For example, to transform the following TRS

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

Submitted to:
HOR 2012

we perform a pattern matching with $\mathscr{P}$ against $\mathscr{R}_{\mathsf{sum}}$. Then a *term homomorphism* φ satisfying $\mathscr{R}_{\mathsf{sum}} = \varphi(\mathscr{P})$ is obtained using a matching algorithm [1]. Now the iterative form of $\mathscr{R}_{\mathsf{sum}}$ is obtained as

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

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

$$\mathscr{E}_{\mathsf{sum}} = \left\{ \; +(0,x_1) \approx +(x_1,0), \quad +(+(x_2,v_2),w_2) \approx +(x_2,+(v_2,w_2)) \; \right\}$$

These equations are used at the verification of the correctness of transformation not only in the framework based on term rewriting but also the one based on lambda calculus. We showed that the correctness of transformation may be (partly) verified by proving properties of term rewriting systems for suitable templates [1, 2]. All recursive programs must be described as program schemes in the framework of lambda calculus in order to use fixed point combinator. In contrast, they can be defined by using case splitting in our framework. Because of universally quantified nature of variables in rewrite rules, second-order matching algorithms on lambda term are not directly applicable in our setting [1].

Because this framework is based on the first order term rewriting, it is difficult to deal with higher order programs. In this paper, we extend the framework of program transformation by templates on first order term rewriting to *simply typed term rewriting* [6], which is one of the simplest frameworks of higher order term rewriting. Our key idea is introducing two kinds of function application, one for function application on STTRSs and one for applying second order matching solution.

The first question to develop such a framework is which term language presenting program schemas to chose. We first introduce a set of *basic types* B and a set *type variables* U. The idea is that in program templates types are not fixed and type variables contained in templates are instantiated at the time of concrete transformations. We call simple types over $B \cup U$ *type pattern* and refer *types* those over B. Next we assume that each *constant* and *local variable* are associated with its type and type pattern, respectively and each *pattern variable* is associated with its argument type patterns and result type pattern. The set of pattern variables whose argument type patterns are $\tau_1,\ldots,\tau_n$ and result type patterns are τ is denoted as $\mathscr{X}^{\tau_1\times\cdots\times\tau_n\Rightarrow\tau}$.

Definition 1 (Term pattern) *The set* $\mathrm{T}^{\tau}(\Sigma,\mathscr{X},\mathscr{V})$ *of* term pattern of type pattern τ *over constants* $\Sigma = \bigcup_{\tau\in\mathrm{ST}(B)}\Sigma^{\tau}$, *pattern variable* $\mathscr{X} = \bigcup_{\tau,\tau_1,\ldots,\tau_n\in\mathrm{ST}(B,U)}\mathscr{X}^{\tau_1\times\cdots\times\tau_n\Rightarrow\tau}$ *and local variables* $\mathscr{V} = \bigcup_{\tau\in\mathrm{ST}(B,U)}\mathscr{V}^{\tau}$ *is defined as: (1)* $\Sigma^{\tau}\cup\mathscr{V}^{\tau} \subseteq \mathrm{T}^{\tau}(\Sigma,\mathscr{X},\mathscr{V})$, *(2)* $s \in \mathrm{T}^{\tau_1\times\cdots\times\tau_n\to\tau}(\Sigma,\mathscr{X},\mathscr{V})$ $(n\geq 1)$ *and* $t_i \in \mathrm{T}^{\tau_i}(\Sigma,\mathscr{X},\mathscr{V})$ *for all* $i\in\{1,\ldots,n\}$ *imply* $(s\ t_1\ \cdots\ t_n) \in \mathrm{T}^{\tau}(\Sigma,\mathscr{X},\mathscr{V})$, *and (3)* $p \in \mathscr{X}^{\tau_1\times\cdots\times\tau_n\Rightarrow\tau}$ $(n\geq 0)$ *and* $t_i \in \mathrm{T}^{\tau_i}(\Sigma,\mathscr{X},\mathscr{V})$ *for all* $i\in\{1,\ldots,n\}$ *imply* $p\langle t_1,\ldots,t_n\rangle \in \mathrm{T}^{\tau}(\Sigma,\mathscr{X},\mathscr{V})$.

Here note that two kinds of function application are introduced—one is $(s\ t_1\ \cdots\ t_n)$ for the function application on object language (simply typed terms) and the other is $p\langle t_1,\ldots,t_n\rangle$ for the function application to be used instantiating program templates to concrete programs. This is contrast to the second-order matching frameworks on lambda terms where these two kinds of function application are identified.

Let $B = \{\mathsf{Nat},\mathsf{List}\}$, $U = \{\alpha,\beta,\gamma,\delta,\varepsilon\}$, $\Sigma = \{\,[\,]^{\mathsf{List}}, :^{\mathsf{Nat}\times\mathsf{List}\to\mathsf{List}}, @^{\mathsf{List}\times\mathsf{List}\to\mathsf{List}}, \mathsf{map}^{(\mathsf{Nat}\to\mathsf{Nat})\times\mathsf{List}\to\mathsf{List}}, \mathsf{mapapp}^{(\mathsf{Nat}\to\mathsf{Nat})\times\mathsf{List}\times\mathsf{List}\to\mathsf{List}}\,\}$, and $\mathscr{X} = \{\mathsf{a}^{\Rightarrow\alpha}, \mathsf{b}^{\gamma\Rightarrow\delta}, \mathsf{c}^{\varepsilon\times\alpha\Rightarrow\alpha}, \mathsf{d}^{\varepsilon\Rightarrow\varepsilon}, \mathsf{e}^{\varepsilon\times\gamma\times\delta\Rightarrow\delta}, \mathsf{f}^{\alpha\times\beta\times\gamma\Rightarrow\delta},$

$$\textbf{Constant}\quad [A \trianglelefteq A, \sigma] \vdash \sigma \qquad \textbf{Var1}\quad [\alpha \trianglelefteq \tau', \sigma] \vdash \sigma \quad \text{if } \sigma(\alpha) = \tau'$$

$$\textbf{Var2}\quad [\alpha \trianglelefteq \tau', \sigma] \vdash \{\alpha := \tau'\} \circ \sigma \quad \text{if } \alpha \notin \mathrm{dom}(\sigma)$$

$$\textbf{Function}\quad \frac{[\tau_1 \trianglelefteq \tau_1', \sigma] \vdash \sigma' \quad [\tau_2 \times \cdots \times \tau_n \to \tau \trianglelefteq \tau_2' \times \cdots \times \tau_n' \to \tau', \sigma'] \vdash \sigma''}{[\tau_1 \times \tau_2 \times \cdots \times \tau_n \to \tau \trianglelefteq \tau_1' \times \tau_2' \times \cdots \times \tau_n' \to \tau', \sigma] \vdash \sigma''}$$

Figure 1: Type matching rules

$\mathsf{g}^{\alpha\times\gamma\Rightarrow\delta}, \mathsf{h}^{\alpha\times\beta\Rightarrow\alpha}, \mathsf{r}^{\beta\Rightarrow\alpha}\}$. An example of program transformation template is like this:

$$\mathscr{P} = \left\{ \begin{array}{lcl} \mathsf{f}\langle u,v,w\rangle & \to & \mathsf{g}\langle \mathsf{h}\langle u,v\rangle, w\rangle \\ \mathsf{g}\langle \mathsf{a}\langle\rangle, u\rangle & \to & \mathsf{b}\langle u\rangle \\ \mathsf{g}\langle \mathsf{c}\langle u,v\rangle, w\rangle & \to & \mathsf{e}\langle u, w, \mathsf{g}\langle v,w\rangle\rangle \\ \mathsf{h}\langle \mathsf{a}\langle\rangle, u\rangle & \to & \mathsf{r}\langle u\rangle \\ \mathsf{h}\langle \mathsf{c}\langle u,v\rangle, w\rangle & \to & \mathsf{c}\langle \mathsf{d}\langle u\rangle, \mathsf{h}\langle v,w\rangle\rangle \end{array} \right\}, \mathscr{P}' = \left\{ \begin{array}{lcl} \mathsf{f}\langle \mathsf{a}\langle\rangle, v, w\rangle & \to & \mathsf{g}\langle \mathsf{r}\langle v\rangle, w\rangle \\ \mathsf{f}\langle \mathsf{c}\langle u,v\rangle, w, z\rangle & \to & \mathsf{e}\langle \mathsf{d}\langle u\rangle, z, \mathsf{f}\langle v,w,z\rangle\rangle \\ \mathsf{g}\langle \mathsf{a}\langle\rangle, u\rangle & \to & \mathsf{b}\langle u\rangle \\ \mathsf{g}\langle \mathsf{c}\langle u,v\rangle, w\rangle & \to & \mathsf{e}\langle u, w, \mathsf{g}\langle v,w\rangle\rangle \\ \mathsf{h}\langle \mathsf{a}\langle\rangle, u\rangle & \to & \mathsf{r}\langle u\rangle \\ \mathsf{h}\langle \mathsf{c}\langle u,v\rangle, w\rangle & \to & \mathsf{c}\langle \mathsf{d}\langle u\rangle, \mathsf{h}\langle v,w\rangle\rangle \end{array} \right\}, \mathscr{H} = \emptyset$$

This transformation template can be used to transform simply typed term rewriting systems (STTRSs):

$$\left\{ \begin{array}{lcl} \mathsf{mapapp}\ f\ xs\ ys & \to & \mathsf{map}\ f\ (@\ xs\ ys) \\ \mathsf{map}\ f\ [\,] & \to & [\,] \\ \mathsf{map}\ f\ (:x\ xs) & \to & :(f\ x)\ (\mathsf{map}\ f\ xs) \\ @\ [\,]\ ys & \to & ys \\ @\ (:x\ xs)\ ys & \to & :x\ (@\ xs\ ys) \end{array} \right\} \Rightarrow \left\{ \begin{array}{lcl} \mathsf{mapapp}\ f\ [\,]\ ys & \to & \mathsf{map}\ f\ ys \\ \mathsf{mapapp}\ f\ (:x\ xs)\ ys & \to & :(f\ x)\ (\mathsf{mapapp}\ f\ xs\ ys) \\ \mathsf{map}\ f\ [\,] & \to & [\,] \\ \mathsf{map}\ f\ (:x\ xs) & \to & :(f\ x)\ (\mathsf{map}\ f\ xs) \\ @\ [\,]\ ys & \to & ys \\ @\ (:x\ xs)\ ys & \to & :x\ (@\ xs\ ys) \end{array} \right\}.$$

One may wonder why there exist common rules in input and output which seem unnecessary for specifying the transformation. They, however, are prepared to guarantee the correctness of the transformation. We note that transformations can be applied even if additional rules (other than target rules) are involved.

In the rest of the paper, we show how such a transformation by templates on STTRSs can be done.

2 Pattern matching algorithm

In this section, we present our pattern matching algorithm. Since term patterns possibly contains type variables, type matching is performed during the pattern matching algorithm on the fly to obtain type consistent term homomorphisms. Type matching can be done as in the first order matching except that it has to be checked incrementally. A *type substitution* is a mapping from type variables to type patterns.

Definition 2 (Type matching) *Let τ be a type pattern, τ' a type and σ and σ' type substitutions. We write $[\tau \trianglelefteq \tau', \sigma] \vdash \sigma'$ if there is an inference by applying the rules listed in Figure 1.*

A *context* is a term containing *holes* $\Box_i$. $\mathrm{C}_n(\Sigma)$ is the set of contexts containing only holes $\Box_1, \ldots, \Box_n$. $C\langle s_1, \ldots, s_n\rangle$ is the result of a context $C \in \mathrm{C}_n(\Sigma)$ replacing $\Box_i$ by s_i. We write $C \in \mathrm{C}^{\tau_1\times\cdots\times\tau_n\Rightarrow\tau}(\Sigma)$ if $\Box_i^{\tau_i}$ for all $\Box_i \in C$ and C^{τ}. A *term homomorphism* is a mapping φ from $\mathscr{V} \cup \mathscr{X}$ to $\mathscr{V} \cup \mathrm{C}(\Sigma)$ satisfying: (1) the restriction of φ on $\mathscr{V}$ is an injective mapping from $\mathrm{dom}_{\mathscr{V}}(\varphi) = \{x \in \mathscr{V} \mid \varphi(x) \neq x\}$ to $\mathscr{V}$ and (2) $\varphi(p) \in \mathrm{C}_n(\Sigma)$ for any $p \in \mathscr{X}$, where $n = \mathrm{arity}(p)$. We put $\mathrm{dom}_{\mathscr{X}}(\varphi) = \{p \in \mathscr{X} \mid \varphi(p) \neq p\langle\Box_1, \ldots, \Box_{\mathrm{arity}(p)}\rangle\}$.

Bound $\dfrac{\langle S \cup \{x^\tau \trianglelefteq y^{\tau'}\}, \sigma, \varphi\rangle}{\langle S, \sigma', \varphi \cup \{x \mapsto y\}\rangle}$ if $\varphi(x) = y \wedge \sigma' = \sigma$, or $x \notin \mathrm{dom}(\varphi) \wedge y \notin \mathrm{range}(\varphi) \wedge [\tau \trianglelefteq \tau', \sigma] \vdash \sigma'$

Remove $\dfrac{\langle S \cup \{f^\tau \trianglelefteq f^\tau\}, \sigma, \varphi\rangle}{\langle S, \sigma, \varphi\rangle}$

Split $\dfrac{\langle S \cup \{(s_1\ \cdots\ s_n)^\tau \trianglelefteq (t_1\ \cdots\ t_n)^{\tau'}\}, \sigma, \varphi\rangle}{\langle S \cup \{s_i \trianglelefteq t_i \mid 1 \le i \le n\}, \sigma', \varphi\rangle}$ if $[\tau \trianglelefteq \tau', \sigma] \vdash \sigma'$

Extend $\dfrac{\langle S \cup \{p\langle s_1, \ldots, s_n\rangle^\tau \trianglelefteq C\langle t_1, \ldots, t_n\rangle^{\tau'}\}, \sigma, \varphi\rangle}{\langle \{p \mapsto C\}(S \cup \{s_i \trianglelefteq t_i \mid \Box_i \in C\}), \sigma', \varphi \cup \{p \mapsto C\}\rangle}$ if $[\tau \trianglelefteq \tau', \sigma] \vdash \sigma'$

Figure 2: Rules of pattern matching algorithm

$$
\begin{aligned}
&\langle \{\mathsf{g}\langle \mathsf{c}\langle u, v\rangle, w\rangle \trianglelefteq \mathsf{map}\ f\ (:x\ xs),\ \mathsf{e}\langle u, w, \mathsf{g}\langle v, w\rangle\rangle \trianglelefteq (f\ x)\ (\mathsf{map}\ f\ xs)\}, \emptyset, \emptyset\rangle \\
\Longrightarrow\ &\langle \{w \trianglelefteq f, \mathsf{c}\langle u, v\rangle \trianglelefteq x\ xs,\ \mathsf{e}\langle u, w, (\mathsf{map}\ w\ v)\rangle \trianglelefteq (f\ x)\ (\mathsf{map}\ f\ xs)\}, \{\delta := \mathsf{List}\}, \{\mathsf{g} \mapsto \mathsf{map}\ \Box_2\ \Box_1\}\rangle \\
\Longrightarrow\ &\langle \{\mathsf{c}\langle u, v\rangle \trianglelefteq x\ xs,\ \mathsf{e}\langle u, w, (\mathsf{map}\ w\ v)\rangle \trianglelefteq (f\ x)\ (\mathsf{map}\ f\ xs)\}, \left\{\begin{array}{l}\delta := \mathsf{List},\\ \gamma := \mathsf{Nat} \to \mathsf{Nat}\end{array}\right\}, \left\{\begin{array}{l}w \mapsto f,\\ \mathsf{g} \mapsto \mathsf{map}\ \Box_2\ \Box_1\end{array}\right\}\rangle \\
\Longrightarrow\ &\langle \{u \trianglelefteq x, v \trianglelefteq xs, \mathsf{e}\langle u, w, (\mathsf{map}\ w\ v)\rangle \trianglelefteq (f\ x)\ (\mathsf{map}\ f\ xs)\}, \left\{\begin{array}{l}\delta := \mathsf{List},\\ \gamma := \mathsf{Nat} \to \mathsf{Nat}\\ \alpha := \mathsf{List}\end{array}\right\}, \left\{\begin{array}{l}w \mapsto f,\\ \mathsf{g} \mapsto \mathsf{map}\ \Box_2\ \Box_1\\ \mathsf{c} \mapsto : \Box_1\ \Box_2\end{array}\right\}\rangle \\
\Longrightarrow\ &\langle \{v \trianglelefteq xs, \mathsf{e}\langle u, w, (\mathsf{map}\ w\ v)\rangle \trianglelefteq (f\ x)\ (\mathsf{map}\ f\ xs)\}, \left\{\begin{array}{lll}\delta := \mathsf{List}, & \gamma := \mathsf{Nat} \to \mathsf{Nat}\\ \alpha := \mathsf{List}, & \varepsilon := \mathsf{Nat}\end{array}\right\}, \left\{\begin{array}{lll}w \mapsto f, & & \mathsf{g} \mapsto \mathsf{map}\ \Box_2\ \Box_1\\ \mathsf{c} \mapsto : \Box_1\ \Box_2, & & u \mapsto x\end{array}\right\}\rangle \\
\Longrightarrow\ &\langle \{\mathsf{e}\langle u, w, (\mathsf{map}\ w\ v)\rangle \trianglelefteq (f\ x)\ (\mathsf{map}\ f\ xs)\}, \left\{\begin{array}{lll}\delta := \mathsf{List}, & \gamma := \mathsf{Nat} \to \mathsf{Nat}\\ \alpha := \mathsf{List}, & \varepsilon := \mathsf{Nat}\end{array}\right\}, \left\{\begin{array}{lll}w \mapsto f, & & \mathsf{g} \mapsto \mathsf{map}\ \Box_2\ \Box_1\\ \mathsf{c} \mapsto : \Box_1\ \Box_2, & & u \mapsto x\\ v \mapsto xs\end{array}\right\}\rangle \\
\Longrightarrow\ &\langle \{u \trianglelefteq x, w \trianglelefteq f, \mathsf{map}\ w\ v \trianglelefteq \mathsf{map}\ f\ xs\}, \left\{\begin{array}{lll}\delta := \mathsf{List}, & \gamma := \mathsf{Nat} \to \mathsf{Nat}\\ \alpha := \mathsf{List}, & \varepsilon := \mathsf{Nat}\end{array}\right\}, \left\{\begin{array}{lll}w \mapsto f, & & \mathsf{g} \mapsto \mathsf{map}\ \Box_2\ \Box_1\\ \mathsf{c} \mapsto : \Box_1\ \Box_2, & & u \mapsto x\\ v \mapsto xs, & & \mathsf{e} \mapsto : (\Box_2\ \Box_1)\ \Box_3\end{array}\right\}\rangle \\
\overset{*}{\Longrightarrow}\ &\langle \emptyset, \left\{\begin{array}{lll}\delta := \mathsf{List}, & \gamma := \mathsf{Nat} \to \mathsf{Nat}\\ \alpha := \mathsf{List}, & \varepsilon := \mathsf{Nat}\end{array}\right\}, \left\{\begin{array}{lll}w \mapsto f, & \mathsf{g} \mapsto \mathsf{map}\ \Box_2\ \Box_1, & \mathsf{c} \mapsto : \Box_1\ \Box_2\\ u \mapsto x, & v \mapsto xs, & \mathsf{e} \mapsto : (\Box_2\ \Box_1)\ \Box_3\end{array}\right\}\rangle
\end{aligned}
$$

Figure 3: Derivation of **ST-Match**

φ is *consistent with* a type substitution σ if (1) $\varphi(x) \in \mathscr{V}^{\sigma(\tau)}$ for any $x \in \mathscr{V}^\tau \cap \mathrm{dom}_{\mathscr{V}}(\varphi)$ and (2) $\varphi(p) \in \mathrm{C}^{\sigma(\tau_1)\times\cdots\times\sigma(\tau_n)\Rightarrow\sigma(\tau)}(\Sigma)$ for any $p \in \mathscr{X}^{\tau_1\times\cdots\times\tau_n\Rightarrow\tau} \cap \mathrm{dom}_{\mathscr{X}}(\varphi)$.

A *matching pair* $s \trianglelefteq t$ is a pair of a term pattern s and a term t, and a *matching problem* is a finite set of matching pairs. Given a matching problem S, our pattern matching algorithm solves whether there exist term homomorphism φ and type substitution σ such that φ is consistent with σ, and $\varphi(s) = t$ holds for any $s \trianglelefteq t \in S$. Our algorithm is given by inference rules acting on a *configuration* $\langle S, \sigma, \varphi\rangle$.

Definition 3 (ST-Match) *Let $\Longrightarrow$ be a relation on configurations defined by: $\langle S, \sigma, \varphi\rangle \Longrightarrow \langle S', \sigma', \varphi'\rangle$ if $\langle S, \sigma, \varphi\rangle$ is rewritten to $\langle S', \sigma', \varphi'\rangle$ by applying the rules listed in Figure 2. The reflexive transitive closure of $\Longrightarrow$ is denoted by $\overset{*}{\Longrightarrow}$. The procedure **ST-Match** is given as follows*

Input: *a matching problem S*

Output: *a pair $\langle \varphi, \sigma\rangle$ of a term homomorphism φ and a type substitution σ if $\langle S, \emptyset, \emptyset\rangle \overset{*}{\Longrightarrow} \langle \emptyset, \sigma, \varphi\rangle$.*

Note that **Extend** rule select an appropriate context *non-deterministically*—our algorithm intends conciseness but not to describe how it can be implemented efficiently. In Figure 3, we present a derivation of **ST-Match** for a pattern matching problem involved in the program transformation in Section 1.

Theorem 4 (Termination of ST-Match) ***ST-Match*** *terminates for any input.*

$$\langle \varphi, \sigma \rangle \vdash x^{\tau} \trianglelefteq y^{\tau'} \quad \text{if } \varphi(x) = y \wedge \sigma(\tau) = \tau' \qquad \frac{\langle \varphi, \sigma \rangle \vdash s_1 \trianglelefteq t_1 \quad \cdots \quad \langle \varphi, \sigma \rangle \vdash s_n \trianglelefteq t_n}{\langle \varphi, \sigma \rangle \vdash (s_1 \ \cdots \ s_n)^{\tau} \trianglelefteq (t_1 \ \cdots \ t_n)^{\tau'}} \quad \text{if } \sigma(\tau) = \tau'$$

$$\langle \varphi, \sigma \rangle \vdash c^{\tau} \trianglelefteq c^{\tau} \qquad \frac{\langle \varphi, \sigma \rangle \vdash s_{i_1} \trianglelefteq t_{i_1} \quad \cdots \quad \langle \varphi, \sigma \rangle \vdash s_{i_m} \trianglelefteq t_{i_m}}{\langle \varphi, \sigma \rangle \vdash p\langle s_1, \ldots, s_n \rangle^{\tau} \trianglelefteq C\langle t_1, \ldots, t_n \rangle^{\tau'}} \quad \text{if } \begin{cases} \{i_1, \ldots, i_m\} = \{i \mid \Box_i \in C\}, \\ \sigma(\tau) = \tau', \text{ and } \varphi(p) = C \end{cases}$$

Figure 4: Inference rules for checking solutions of configurations

3 Correctness of pattern matching algorithm

In this section, we show soundness and completeness of **ST-Match** in order to ensure the correctness of the algorithm. We say a pair $\langle \varphi, \sigma \rangle$ of a term homomorphism φ and a type substitution σ is a *solution* of a matching problem S if (1) φ is consistent with σ and (2) $\varphi(s) = t$ for all $s \trianglelefteq t \in S$.

Definition 5 (Solution of configuration) *(a) We write $\langle \varphi, \sigma \rangle \vdash s \trianglelefteq t$ if there is an inference tree by applying the rules in Figure 4.*

(b) A pair $\langle \tilde{\varphi}, \tilde{\sigma} \rangle$ of a term homomorphism $\tilde{\varphi}$ and a type substitution $\tilde{\sigma}$ is a solution *of a configuration $\langle S, \sigma, \varphi \rangle$ if (1) $\langle \varphi, \sigma \rangle \vdash s \trianglelefteq t$ for all $s \trianglelefteq t \in S$, (2) $\sigma \subseteq \tilde{\sigma}$, and (3) $\varphi \subseteq \tilde{\varphi}$.*

For ensuring type consistency of outputs of **ST-Match**, we consider the followings.

Lemma 6 (1) Suppose $\langle S, \sigma, \varphi \rangle \Longrightarrow \langle S', \sigma', \varphi' \rangle$. If $x \in \mathcal{V}^{\tau}$ implies $\varphi(x) \in \mathcal{V}^{\sigma(\tau)}$ for any $x \in \mathrm{dom}_{\mathcal{V}}(\varphi)$, then $x \in \mathcal{V}^{\tau}$ implies $\varphi'(x) \in \mathcal{V}^{\sigma(\tau)}$ for any $x \in \mathrm{dom}_{\mathcal{V}}(\varphi')$. (2) Suppose $\langle \tilde{\varphi}, \tilde{\sigma} \rangle \vdash s \trianglelefteq t$. $p \in \mathcal{X}^{\tau_1 \times \cdots \times \tau_n \Rightarrow \tau}$ implies $\tilde{\varphi}(p) \in \mathrm{C}^{\tilde{\sigma}(\tau_1) \times \cdots \times \tilde{\sigma}(\tau_n) \Rightarrow \tilde{\sigma}(\tau)}(\Sigma)$ for any $p \in \mathcal{X}(s) \cap \mathrm{dom}_{\mathcal{X}}(\tilde{\varphi})$.

Next lemma can be shown by using straightforward case splitting of applied rules.

Lemma 7 *Suppose $\langle S, \sigma, \varphi \rangle \Longrightarrow \langle S', \sigma', \varphi' \rangle$ and $x \in \mathcal{V}^{\tau}$ implies $\varphi(x) \in \mathcal{V}^{\sigma(\tau)}$ for any $x \in \mathrm{dom}_{\mathcal{V}}(\varphi)$. If $\langle \tilde{\varphi}, \tilde{\sigma} \rangle$ is a solution of $\langle S', \sigma', \varphi' \rangle$, then $\langle \tilde{\varphi}, \tilde{\sigma} \rangle$ is also a solution of $\langle S, \sigma, \varphi \rangle$.*

From lemmas above, we obtain the following result of soundness.

Theorem 8 (Soundness of ST-Match) *If **ST-Match** produces $\langle \tilde{\varphi}, \tilde{\sigma} \rangle$ for an input S, then $\langle \tilde{\varphi}, \tilde{\sigma} \rangle$ is a solution of S.*

In order to show completeness of **ST-Match**, we show the following lemmas.

Lemma 9 *Let $\langle S, \sigma, \varphi \rangle$ be a configuration with $S \neq \emptyset$. If $\langle \tilde{\varphi}, \tilde{\sigma} \rangle$ is a solution of $\langle S, \sigma, \varphi \rangle$, then there exists a solution $\langle S', \sigma', \varphi' \rangle$ such that (1) $\langle S, \sigma, \varphi \rangle \Longrightarrow \langle S', \sigma', \varphi' \rangle$ and (2) $\langle \tilde{\varphi}, \tilde{\sigma} \rangle$ is a solution of $\langle S', \sigma', \varphi' \rangle$.*

Lemma 10 *Let $\langle \tilde{\varphi}, \tilde{\sigma} \rangle$ be a solution of a configuration $\langle S, \sigma, \varphi \rangle$. If there exist a term homomorphism φ' and a type substitution σ' such that $\langle S, \sigma, \varphi \rangle \Longrightarrow \langle \emptyset, \sigma', \varphi' \rangle$, then (1) $\sigma' \subseteq \tilde{\sigma}$, and (2) $\varphi' \subseteq \tilde{\varphi}$.*

We now obtain the following theorem from lemmas above.

Theorem 11 (Completeness of ST-Match) *Let Φ be the set of outputs of **ST-Match** for input matching problem S. If $\langle \varphi, \sigma \rangle$ is a solution of S, then there exists $\langle \tilde{\varphi}, \tilde{\sigma} \rangle \in \Phi$ such that $\tilde{\varphi} \subseteq \varphi$ and $\tilde{\sigma} \subseteq \sigma$.*

References

[1] Y. Chiba, T. Aoto & Y. Toyama (2005): *Program transformation by templates based on term rewriting*. In: *Proc. of PPDP 2005*, ACM Press, pp. 59–69.

[2] Y. Chiba, T. Aoto & Y. Toyama (2010): *Program transformation templates for tupling based on term rewriting*. *IEICE Transactions* 93-D(5), pp. 963–973.

[3] R. Curien, Z. Qian & H. Shi (1996): *Efficient second-order matching*. In: *Proc. of RTA 1996*, *LNCS* 1103, Springer-Verlag, pp. 317–331.

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

[5] O. de Moor & G. Sittampalam (2001): *Higher-order matching for program transformation*. *TCS* 269(1–2), pp. 135–162.

[6] T. Yamada (2001): *Confluence and termination of simply typed term rewriting systems*. In: *Proc. of RTA 2001*, *LNCS* 2051, Springer-Verlag, pp. 338–352.

[7] T. Yokoyama, Z. Hu & M. Takeichi (2004): *Deterministic second-order patterns*. *IPL* 89(6), pp. 309–314.