RAPT: A Program Transformation System based on Term Rewriting

Yuki Chiba and Takahito Aoto

Research Institute of Electrical Communication, Tohoku University, Japan
{chiba, aoto}@nue.riec.tohoku.ac.jp

Abstract. Chiba et al. (2005) proposed a framework of program transformation by template based on term rewriting in which correctness of the transformation is verified automatically. This paper describes RAPT (Rewriting-based Automated Program Transformation system) which implements this framework.

1 Introduction

Chiba et al. [4] proposed a framework of program transformation by template based on term rewriting in which correctness of the transformation is verified automatically. In their framework, programs and program schemas are given by term rewriting systems (TRS, for short) and TRS patterns. A program transformation template consists of input and output TRS patterns and a hypothesis which is a set of equations the input TRS has to satisfy to guarantee the correctness of transformation.

This paper describes RAPT (Rewriting-based Automated Program Transformation system) which implements this framework. RAPT transforms a many-sorted TRS according to a specified program transformation template. Based on the rewriting induction proposed by Reddy [14], RAPT automatically verifies whether the input TRS satisfies the hypothesis of the transformation template. It also verifies conditions imposed to the input TRS and generated TRS by utilizing standard techniques in term rewriting. Thus, presupposing the program transformation template is *developed* [4], the correctness of the transformation is automatically verified so that the transformation keeps the relationship between initial ground terms and their normal forms.

2 Transformation by templates

Let $\mathscr{P}$ be a set of (arity-fixed) pattern variables (disjoint from the set $\mathscr{F}$ of function symbols and the set $\mathscr{V}$ of variables). A *pattern* is a term with pattern variables. A *TRS pattern* $\mathcal{P}$ is a set of rewriting rules over patterns. A *hypothesis* $\mathcal{H}$ is a set of equations over patterns. A *transformation template* (or just *template*) is a triple $\langle \mathcal{P}, \mathcal{P}', \mathcal{H} \rangle$ of two TRS patterns $\mathcal{P}$, $\mathcal{P}'$ and a hypothesis $\mathcal{H}$.

The following template $\langle \mathcal{P}, \mathcal{P}', \mathcal{H} \rangle$ describes a well-known transformation from the recursive form to the iterative (tail-recursive) form:

$$\mathcal{P} \begin{cases} \mathsf{f}(\mathsf{a}) & \to \mathsf{b} \\ \mathsf{f}(\mathsf{c}(u,v)) & \to \mathsf{g}(\mathsf{e}(u), \mathsf{f}(v)) \\ \mathsf{g}(\mathsf{b}, u) & \to u \\ \mathsf{g}(\mathsf{d}(u,v), w) & \to \mathsf{d}(u, \mathsf{g}(v,w)) \end{cases}$$

$$\mathcal{P}' \begin{cases} \mathsf{f}(u) & \to \mathsf{f}_1(u, \mathsf{b}) \\ \mathsf{f}_1(\mathsf{a}, u) & \to u \\ \mathsf{f}_1(\mathsf{c}(u,v), w) & \to \mathsf{f}_1(v, \mathsf{g}(w, \mathsf{e}(u))) \\ \mathsf{g}(\mathsf{b}, u) & \to u \\ \mathsf{g}(\mathsf{d}(u,v), w) & \to \mathsf{d}(u, \mathsf{g}(v,w)) \end{cases}$$

$$\mathcal{H} \begin{cases} \mathsf{g}(\mathsf{b}, u) & \approx \mathsf{g}(u, \mathsf{b}) \\ \mathsf{g}(\mathsf{g}(u,v), w) & \approx \mathsf{g}(u, \mathsf{g}(v,w)) \end{cases}$$

Here, the symbols $\mathsf{f}, \mathsf{a}, \mathsf{b}, \mathsf{g}, \ldots$ are pattern variables.

To achieve the program transformation by templates, we need a mechanism to specify how a template is applied to a concrete TRS. For this we use a notion of *term homomorphism* [4]. If we match the TRS pattern $\mathcal{P}$ to a concrete TRS $\mathcal{R}$ with a term homomorphism φ, we obtain a generated TRS $\mathcal{R}'$ by applying φ to the TRS pattern $\mathcal{P}'$ (Figure 1). A matching algorithm to find all (most general) term homomorphisms φ satisfying $\mathcal{R} = \varphi(\mathcal{P})$ from a given TRS $\mathcal{R}$ and a TRS pattern $\mathcal{P}$ is presented in [4].

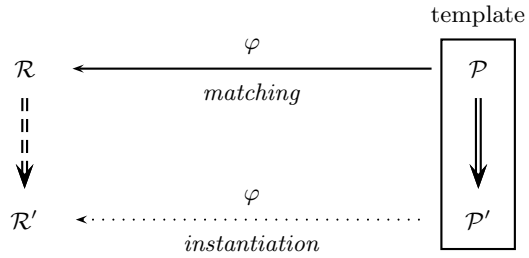

Fig. 1. TRS transformation

Definition 1 ([4]). *Let $\langle \mathcal{P}, \mathcal{P}', \mathcal{H} \rangle$ be a template. A TRS $\mathcal{R}$* is transformed into *$\mathcal{R}'$ by $\langle \mathcal{P}, \mathcal{P}', \mathcal{H} \rangle$ if there exists a term homomorphism φ such that $\mathcal{R} = \varphi(\mathcal{P})$ and $\mathcal{R}' = \varphi(\mathcal{P}')$.*

The following TRS $\mathcal{R}_{sum}$ computes the summation of a list using a recursive call.

$$\mathcal{R}_{sum} \begin{cases} \mathsf{sum}([\,]) & \to \mathsf{0} \\ \mathsf{sum}(x : y) & \to +(x, \mathsf{sum}(y)) \\ +(\mathsf{0}, x) & \to x \\ +(\mathsf{s}(x), y) & \to \mathsf{s}(+(x,y)) \end{cases}$$

The following term homomorphism φ is used to transform the TRS $\mathcal{R}_{sum}$.

$$\varphi = \left\{ \begin{array}{ll} \mathsf{f} \ \mapsto \mathsf{sum}(\Box_1), & \mathsf{b} \mapsto \mathsf{0}, \\ \mathsf{g} \ \mapsto +(\Box_1, \Box_2), & \mathsf{c} \mapsto \Box_1{:}\Box_2, \\ \mathsf{f}_1 \mapsto \mathsf{sum1}(\Box_1, \Box_2), & \mathsf{d} \mapsto \mathsf{s}(\Box_2), \\ \mathsf{a} \ \mapsto [\,], & \mathsf{e} \mapsto \Box_1 \end{array} \right\}$$

Applying φ to $\mathcal{P}'$, we get the following output TRS $\mathcal{R}'_{sum}$.

$$\mathcal{R}'_{sum} \left\{ \begin{array}{ll} \mathsf{sum}(x) & \to \mathsf{sum1}(x, \mathsf{0}) \\ \mathsf{sum1}([\,], x) & \to x \\ \mathsf{sum1}(x : y, z) & \to \mathsf{sum1}(y, +(z, x)) \\ +(\mathsf{0}, x) & \to x \\ +(\mathsf{s}(x), y) & \to \mathsf{s}(+(x, y)) \end{array} \right.$$

$\mathcal{R}'_{sum}$ computes the summation of a list more efficiently without the recursion.

3 Design of RAPT

We assume that the set $\mathscr{F}$ of function symbols is divided into disjoint two sets: the set $\mathscr{F}_{\mathrm{d}}$ of *defined function symbols* and the set $\mathscr{F}_{\mathrm{c}}$ of *constructor symbols*. The following is a sufficient condition to guarantee the correctness of the transformation from a TRS $\mathcal{R}$ on $\mathscr{G}$ to a TRS $\mathcal{R}'$ on $\mathscr{G}'$ by a template $\langle \mathcal{P}, \mathcal{P}', \mathcal{H} \rangle$ through a term homomorphism φ (Theorem 2 of [4]):

- $\mathcal{R}$ is a left-linear confluent constructor system,
- $\langle \mathcal{P}, \mathcal{P}', \mathcal{H} \rangle$ is a developed template,
- φ is a CS-homomorphism,
- equations in $\varphi(\mathcal{H})$ are inductive consequences of $\mathcal{R}$ for $\mathscr{G}$,
- $\mathcal{R}$ is sufficiently complete for $\mathscr{G}$, and
- $\mathcal{R}'$ is sufficiently complete for $\mathscr{G}'$,

where $\mathscr{F}_{\mathrm{c}} \subseteq \mathscr{G}, \mathscr{G}' \subseteq \mathscr{F}$.

A key property of our framework is sufficient completeness, which has to be satisfied by input and output TRSs [4]. Sufficient completeness is checked in RAPT by the decidable necessary and sufficient condition for terminating TRSs [9, 11], and thus currently the target of program transformation by RAPT is limited to terminating TRSs. A simple procedure to check confluence is also available for terminating TRSs [1].

RAPT uses *rewriting induction* [14], in which termination plays an essential role, to verify that the instantiated hypotheses of transformation template are inductive consequences of the input TRS. Since RAPT handles only terminating TRSs, rewriting induction is integrated keeping the whole system simple. Other inductive proving methods [2, 5] also can be possibly incorporated.

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

4 Implementation

4.1 Specification of input TRS and transformation template

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

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

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

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

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

Fig. 2. Specification of input TRS and transformation template

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

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

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

1. `INPUT`: rewrite rules of $\mathcal{P}$ over patterns,
2. `OUTPUT`: rewrite rules of $\mathcal{P}'$ over patterns,
3. `HYPOTHESIS`: equations of $\mathcal{H}$ over patterns.

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

4.2 Implementation details

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

The TRS transformation and the verification of its correctness are conducted in RAPT in 6 phases. In Figure 3, we describe these phases and dependencies among each phases. Solid arrows represent data flow and dotted arrows explain how information obtained in each phase is used.

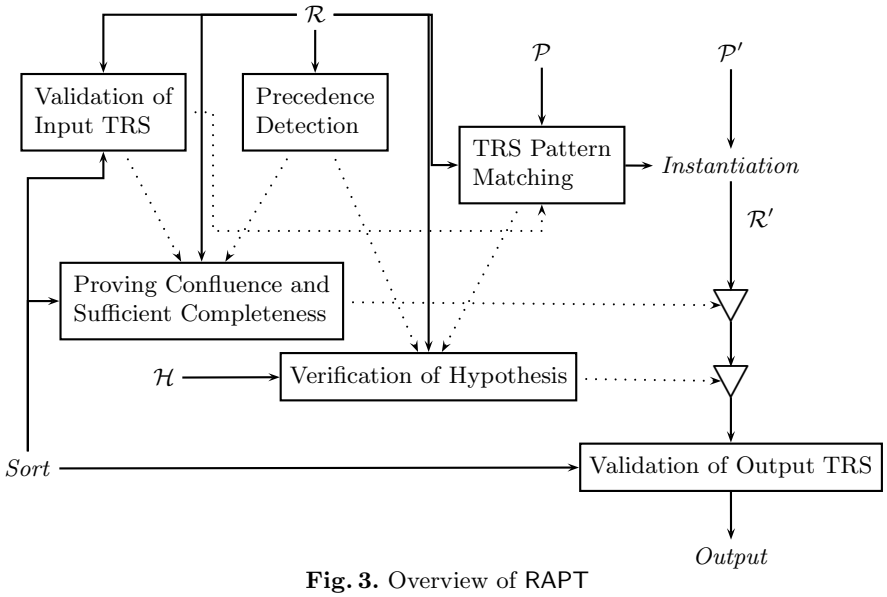

Fig. 3. Overview of RAPT

If these 6 phases are successfully passed then RAPT produces output TRSs. The correctness of the transformation is guaranteed, provided the transformation template is developed. RAPT can also report summaries of program transformation in a readable format (Figure 4).

We now explain operations of each phases briefly.

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

2. Precedence detection In this phase, RAPT checks the input TRS is terminating by LPO and (if it is the case) detects a precedence. The suitable precedence (if there exists one) for LPO is computed based on the LPO constraint solving algorithm described in [7].

3. Proving confluence and sufficient completeness In this phase, RAPT proves whether the input TRS is confluent and sufficiently complete. This makes use of the information of constructor symbols detected at Phase 1 and the fact that the input TRS is left-linear and terminating verified at Phases 1 and 2,

Summary of Program Transformation

reported by RAPT

February 21, 2006

Transformation Template:

$$\mathcal{P}\ \begin{cases} \mathsf{f}(\mathsf{a}) & \to \mathsf{b} \\ \mathsf{f}(\mathsf{c}(u,v)) & \to \mathsf{g}(\mathsf{e}(u,v),\mathsf{f}(v)) \end{cases}$$

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

$$\mathcal{H}\ \begin{cases} \mathsf{g}(\mathsf{b},u) & \approx u \\ \mathsf{g}(u,\mathsf{b}) & \approx u \\ \mathsf{g}(\mathsf{g}(u,v),w) & \approx \mathsf{g}(u,\mathsf{g}(v,w)) \end{cases}$$

Input TRS:

$$\mathcal{R}\begin{cases} \mathsf{rev}(\mathsf{nil}) & \to \mathsf{nil} \\ \mathsf{rev}(\mathsf{cons}(x,ys)) & \to \mathsf{app}(\mathsf{rev}(ys),\mathsf{cons}(x,\mathsf{nil})) \\ \mathsf{app}(\mathsf{nil},x) & \to x \\ \mathsf{app}(\mathsf{cons}(x,y),z) & \to \mathsf{cons}(x,\mathsf{app}(y,z)) \end{cases}$$

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

A solution of matching (CS-homomorphisms):

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

The instantiation of hypothesis:

$$\varphi(\mathcal{H})\begin{cases} \mathsf{app}(u,\mathsf{nil}) & \approx u \\ \mathsf{app}(\mathsf{nil},u) & \approx u \\ \mathsf{app}(w,\mathsf{app}(v,u)) & \approx \mathsf{app}(\mathsf{app}(w,v),u) \end{cases}$$

Output TRS:

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

Fig. 4. Example of a program transformation report

respectively. For confluence, it is checked whether all critical pairs are joinable. For sufficient completeness, quasi-reducibility of the TRS is checked; this part is based on the (many-sorted extension of) complement algorithm introduced in [10] that computes the complement of a substitution.

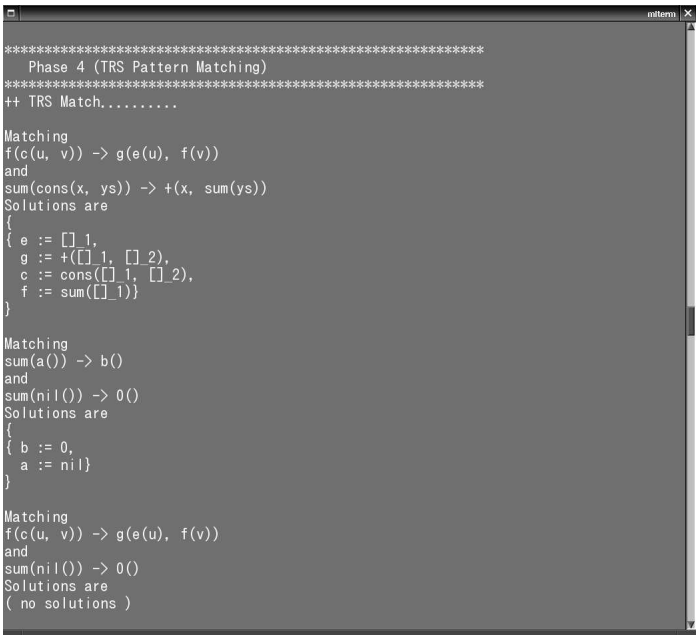

Fig. 5. Snapshot of TRS pattern matching

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

5. Verification of hypothesis In this phase, RAPT checks whether the input TRS satisfies the hypothesis part of the template. This is done by (1) instantiating the hypotheses through the term homomorphism found at Phase 4 and

(2) proving they are inductive consequences of the input TRS, using rewriting induction. The latter uses LPO with the precedence detected at Phase 2.

6. Validation of output TRS In this phase, RAPT checks whether the output TRS is (1) terminating, (2) left-linear, (3) type consistent, and (4) sufficiently complete. In (3), because the pattern TRS $\mathcal{P}'$ for the output may contain a pattern variable not occurring in the pattern TRS $\mathcal{P}$ for the input, types may be unknown for some of function symbols in $\mathcal{R}'$. Therefore, we need to infer the type information together with the type consistency check. (4) is proved based on the fact the output TRS is terminating which is verified at (1) using LPO.

Table 1. Experimental result

Template I	TRSs	Template II	TRSs
$\left\{\begin{array}{ll}\mathsf{f}(\mathsf{a}) & \to \mathsf{b}\\ \mathsf{f}(\mathsf{c}(u,v)) & \to \mathsf{g}(\mathsf{e}(u),\mathsf{f}(v))\\ \mathsf{g}(\mathsf{b},u) & \to u\\ \mathsf{g}(\mathsf{d}(u,v),w) & \to \mathsf{d}(u,\mathsf{g}(v,w))\end{array}\right\}$, $\left\{\begin{array}{ll}\mathsf{f}(u) & \to \mathsf{f}_1(u,\mathsf{b})\\ \mathsf{f}_1(\mathsf{a},u) & \to u\\ \mathsf{f}_1(\mathsf{c}(u,v),w) & \to \mathsf{f}_1(v,\mathsf{g}(w,\mathsf{e}(u)))\\ \mathsf{g}(\mathsf{b},u) & \to u\\ \mathsf{g}(\mathsf{d}(u,v),w) & \to \mathsf{d}(u,\mathsf{g}(v,w))\end{array}\right\}$, $\left\{\begin{array}{ll}\mathsf{g}(\mathsf{b},u) & \approx \mathsf{g}(u,\mathsf{b})\\ \mathsf{g}(\mathsf{g}(u,v),w) & \approx \mathsf{g}(u,\mathsf{g}(v,w))\end{array}\right\}$	3	$\left\{\begin{array}{ll}\mathsf{f}(\mathsf{a}) & \to \mathsf{b}\\ \mathsf{f}(\mathsf{c}(u,v)) & \to \mathsf{g}(\mathsf{f}(v),\mathsf{e}(u))\\ \mathsf{g}(\mathsf{b},u) & \to u\\ \mathsf{g}(\mathsf{d}(u,v),w) & \to \mathsf{d}(u,\mathsf{g}(v,w))\end{array}\right\}$, $\left\{\begin{array}{ll}\mathsf{f}(u) & \to \mathsf{f}_1(u,\mathsf{b})\\ \mathsf{f}_1(\mathsf{a},u) & \to u\\ \mathsf{f}_1(\mathsf{c}(u,v),w) & \to \mathsf{f}_1(v,\mathsf{g}(\mathsf{e}(u),w))\\ \mathsf{g}(\mathsf{b},u) & \to u\\ \mathsf{g}(\mathsf{d}(u,v),w) & \to \mathsf{d}(u,\mathsf{g}(v,w))\end{array}\right\}$, $\left\{\begin{array}{ll}\mathsf{g}(\mathsf{b},u) & \approx \mathsf{g}(u,\mathsf{b})\\ \mathsf{g}(\mathsf{g}(u,v),w) & \approx \mathsf{g}(u,\mathsf{g}(v,w))\end{array}\right\}$	3
Template III	**TRSs**	**Template IV**	**TRSs**
$\left\{\begin{array}{ll}\mathsf{f}(\mathsf{a}) & \to \mathsf{b}\\ \mathsf{f}(\mathsf{c}(u,v)) & \to \mathsf{g}(\mathsf{e}(u,v),\mathsf{f}(v))\end{array}\right\}$, $\left\{\begin{array}{ll}\mathsf{f}(u) & \to \mathsf{f}_1(u,\mathsf{b})\\ \mathsf{f}_1(\mathsf{a},u) & \to u\\ \mathsf{f}_1(\mathsf{c}(u,v),w) & \to \mathsf{f}_1(v,\mathsf{g}(w,\mathsf{e}(u,v)))\end{array}\right\}$, $\left\{\begin{array}{ll}\mathsf{g}(\mathsf{b},u) & \approx u\\ \mathsf{g}(u,\mathsf{b}) & \approx u\\ \mathsf{g}(\mathsf{g}(u,v),w) & \approx \mathsf{g}(u,\mathsf{g}(v,w))\end{array}\right\}$	11	$\left\{\begin{array}{ll}\mathsf{f}(x,y,z) & \to \mathsf{g}(\mathsf{h}(x,y),z)\\ \mathsf{g}(\mathsf{a},y) & \to \mathsf{b}(u)\\ \mathsf{g}(\mathsf{c}(x,y),z) & \to \mathsf{e}(x,\mathsf{g}(y,z))\\ \mathsf{h}(\mathsf{a},y) & \to \mathsf{r}(y)\\ \mathsf{h}(\mathsf{c}(x,y),z) & \to \mathsf{c}(\mathsf{d}(x),\mathsf{h}(y,z))\end{array}\right\}$, $\left\{\begin{array}{ll}\mathsf{f}(\mathsf{a},y,z) & \to \mathsf{g}(\mathsf{r}(y),z)\\ \mathsf{f}(\mathsf{c}(x,y),z,w) & \to \mathsf{e}(\mathsf{d}(x),\mathsf{f}(y,z,w))\\ \mathsf{g}(\mathsf{a},y) & \to \mathsf{b}(u)\\ \mathsf{g}(\mathsf{c}(x,y),z) & \to \mathsf{e}(x,\mathsf{g}(y,z))\\ \mathsf{h}(\mathsf{a},y) & \to \mathsf{r}(y)\\ \mathsf{h}(\mathsf{c}(x,y),z) & \to \mathsf{c}(\mathsf{d}(x),\mathsf{h}(y,z))\end{array}\right\}$, $\{\}$	8

5 Experiments

We have checked operations of RAPT using several templates. Table 1 describes some of transformation templates and numbers of TRSs succeeded in transformation by each template. Template I is the one which appears in Section 2. This template represents a well-known transformation from recursive programs to iterative programs. A same kind of transformation is also described by Template II. The main difference between Template I and II is the right-hand side of second rule of input parts. In our experiments, there exist TRSs which cannot be transformed by one of these templates but can be done by the other. Template

III is the one which overcomes this difference; unchanged rewrite rules of input and output TRS patterns are removed and rewrite rules which are necessary to develop the template are pushed into the hypothesis. Template IV represents another transformation known as fusion or deforestation [16]. RAPT performs transformations of these examples in less than 100 msec.

6 Concluding remarks

Program transformation techniques have been widely investigated in various fields [3, 12, 13, 16]. This paper describes the system RAPT, which implements the program transformation based on term rewriting introduced in [4]. RAPT transforms a term rewriting system according to a specified program transformation template and automatically verifies correctness of the transformation. We have described the design and implementation of RAPT. An experimental result for several templates has been shown.

Another framework of program transformation by templates is the one based on lambda calculus [6, 8, 15]. MAG system [6, 15] is a program transformation system based on this framework. MAG supports transformations which include modification of expressions, matching with a help of hypothesis; its target also includes higher-order programs. RAPT does not handle such refinements, and cannot deal with most of transformations appearing in [15]. The advantage of RAPT against MAG lies on the approach to the verification of hypothesis. Since the correctness of transformation by MAG system is based on Huet and Lang's original framework [8], users are usually need to verify the hypothesis. In contrast, RAPT proves the hypothesis automatically without help of users.

Besides the limitation of the theoretical framework, several limitations are imposed in the current implementation of RAPT:

- RAPT handles only terminating TRSs. In fact, termination of input and output TRSs are not required in the theoretical framework on which RAPT is based. The main reason to limit its target to terminating TRSs is to reduce checking of sufficient completeness to that of quasi-reducibility, which can be easily verified.
- RAPT allows only confluent TRSs for input. Theoretically, not confluence but ground confluence is sufficient. Replacing confluence checking by ground confluence checking might enlarge the scope of input programs.
- RAPT implements only a naive rewriting induction. Thus, incorporating lemma discovery mechanism and other inductive theorem proving methods may largely enhance the power of inductive theorem proving. Since verification of the hypothesis of template is an important part of the correctness verification, enhancing this part will increase the flexibility of the program transformation.

Extending RAPT to make more flexible transformation possible remains as a future work.

Acknowledgments

Thanks are due to anonymous referees for useful comments and advices. The authors also thank Yoshihito Toyama for valuable comments and discussions. This work was partially supported by a grant from Japan Society for the Promotion of Science, No. 17700002.

References

1. F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.
2. A. Bundy. The automation of proof by mathematical induction. In *Handbook of Automated Reasoning*, chapter 13, pages 845–911. Elsevier and MIT Press, 2001.
3. R.M. Burstall and J. Darlington. A transformation system for developing recursive programs. *Journal of the ACM*, 24(1):44–67, 1977.
4. Y. Chiba, T. Aoto, and Y. Toyama. Program transformation by templates based on term rewriting. In *Proceedings of the 7th ACM-SIGPLAN International Conference on Principles and Practice of Declarative Programming (PPDP 2005)*, pages 59–69. ACM Press, 2005.
5. H. Comon. Inductionless induction. In *Handbook of Automated Reasoning*, chapter 14, pages 913–962. Elsevier and MIT Press, 2001.
6. O. de Moor and G. Sittampalam. Higher-order matching for program transformation. *Theoretical Computer Science*, 269:135–162, 2001.
7. N. Hirokawa and A. Middeldorp. Tsukuba termination tool. In *Proceedings of the 14th International Conference on Rewriting Techniques and Applications*, volume 2706 of *LNCS*, pages 311–320. Springer-Verlag, 2003.
8. G. Huet and B. Lang. Proving and applying program transformations expressed with second order patterns. *Acta Informatica*, 11:31–55, 1978.
9. D. Kapur, P. Narendran, and H. Zhang. On sufficient-completeness and related properties of term rewriting systems. *Acta Informatica*, 24(4):395–415, 1987.
10. A. Lazrek, P. Lescanne, and J. J. Thiel. Tools for proving inductive equalities, relative completeness, and ω-completeness. *Information and Computation*, 84:47–70, 1990.
11. T. Nipkow and G. Weikum. A decidability result about sufficient-completeness of axiomatically specified abstract data types. In *Proceedings of the 6th GI-Conference on Theoretical Computer Science*, volume 145 of *LNCS*, pages 257–268. Springer-Verlag, 1983.
12. R. Paige. Future directions in program transformations. *ACM Computing Surveys*, 28(4es):170, 1996.
13. H. Partsch and R. Steinbrüggen. Program transformation systems. *ACM Computing Surveys*, 15(3):199–236, 1983.
14. U. S. Reddy. Term rewriting induction. In *Proceedings of the 10th International Conference on Automated Deduction*, volume 449 of *LNAI*, pages 162–177, 1990.
15. G. Sittampalam. *Higher-Order Matching for Program Transformation*. PhD thesis, Magdalen College, 2001.
16. P. Wadler. Deforestation: transforming programs to eliminate trees. *Theoretical Computer Science*, 73:231–248, 1990.