분리성에 기반한 직교항 개서 시스템의 변환
(Transformation of Orthogonal Term Rewriting Systems based on Separability)

변 석 우 †
(Sugwoo Byun)

요 약 직교항 개서 시스템은 함수형 언어와 그 구현 과정을 잘 반영하고 있다. 본 논문에서는 룰의 왼쪽 부분을 매우 간략한 형태로 된 flat 시스템으로 변역하는 기법에 대해서 논의한다. 직교항 개서 시스템의 한 부류인 transformable OTRSs를 정의하고, 이 시스템들은 flat 시스템으로 변환될 수 있음을 보인다. 이 변환은 람다 계산법에서 연구된 분리성 이론을 기반으로 하고 있는데, 본 논문에서는 직교항 개서 시스템에서의 분리성과 강력 순차성에 대한 연관성에 대해서도 논의하고 있다.

Abstract OTRSs (orthogonal term rewriting systems) provide a well-known description of functional languages and their implementation. This paper describes some ways of transforming such systems into systems having simpler left-hand sides, the goal being to transform them into 'flat' OTRSs, which have a particularly simple form. A class of systems, the transformable OTRSs, which allow such a transformation is defined. Transformation, in this paper, is based on the separability theory, originally developed in the λ-calculus. We also associate separability with strong-sequentiality in OTRSs.

1. Introduction

There is a well-developed theory of *OTRSs* (orthogonal term rewriting systems) [1][2][3][4]. Such systems are well-known as a description of computation in functional languages. The task of pattern-matching in such systems can be rather complex, as demonstrated by [1]. One method of simplifying this task is to transform OTRSs into OTRSs whose rules have simpler left-hand sides. In this paper, we define several such transformations, which together transform many OTRSs into a particularly simple form, the so-called *flat* systems. The class of systems which these transformations can "flatten", the *transformable* systems, lies strictly between the class of strongly sequential constructor systems and the class of strongly sequential systems.

Huet and Lévy developed a foundational theory of pattern-matching semantics known as[1], where pattern-matching sequences of strong sequential systems are described as *matching dags*. In this paper, we define *separation trees* for *constructor systems*, the idea of which is borrowed from 'Böhm separability' in the λ-calculus. Each pattern-matching sequence in *strongly sequential constructor systems* is described as a separation tree. Intuitively the condition of transformation is that a transformed system should preserve a pattern-matching sequence of the original system. Given a TRS R with a separation tree U and a transformation function T, a relation between R and $T(R)$ can be represented by a relation between U and subtrees of U.

Kennaway has introduced some transformation functions in[5]. Here, we define the *generic transformation* which generalizes Kennaway's transformation

† 정 회 원 : 경성대학교 정보과학부 교수
swbyun@star.ks.ac.kr
논문접수 : 2000년 12월 14일
심사완료 : 2001년 6월 13일

functions. Whereas those functions are defined for constructor systems, we would like to consider a TRS which can be transformed into a strongly sequential constructor system. It is difficult for us to show every strongly seqeuntial system as well as orthogonal TRSs can be transformed. We show *transformable systems*, a subclass of strongly sequential systems, can be transformed. The correctness condition is strengthened. Given a system and its transformation, we show that not only a transformed system simulates the original system (forward simulation) but the original system also simulates the transformed system (backward simulation). This means the existence of a bisimulation between two systems.

Transformation techniques based on strong sequentiality are applied to the the implementation of functional programming languages[6] and a term graph rewriting language Dactl[5]. Unless it is mentioned, TRSs mean orthogonal TRSs in this paper.

2. Preliminaries

A TRS over signature Σ is a pair $(Ter(\Sigma), R)$ consisting of the set $Ter(\Sigma)$ of terms over the signature Σ and a set of rewrite rules $R \subseteq Ter(\Sigma) \times Ter(\Sigma)$. The signature Σ consists of:

- a countably infinite set Var of variables denoted as $x, y, z, ...$,
- a non-empty set of function symbols denoted as capital characters $F, G, ...$, each equipped with arity (a natural number). Function symbols with arity 0 are called *constants*.

The set $Ter(\Sigma)$ of terms over a signature Σ is defined inductively:

- variables $x, y, z, ... \in Ter(\Sigma)$,
- If F is an n-ary function symbol and $t_1, ..., t_n \in Ter(\Sigma)$, then $F(t_1, ..., t_n) \in Ter(\Sigma)$.

The set R of rewrite rules contains pairs (l, r) of terms in $Ter(\Sigma)$, written as $l \to r$, such that

- the left-hand side (or LHS) l is not a variable,
- every variable occurring in the right-hand side (or RHS) r also occurs in l.

If the principal function symbol of l is F, then $l \to r$ is called a *rule for* F, or an *F-rule*. $s \equiv t$ indicates the identity of two terms s and t. The set $O(t)$ of *occurrences* (or *positions*) of a term $t \in Ter(\Sigma)$ is defined by induction on the structure of t as follows: $O(t) = \{\lambda\}$ if t is a variable, and $O(t) = \{\lambda\} \cup \{i \cdot u \mid 1 \le i \le n \text{ and } u \in O(t_i)\}$, if t is of the form $F(t_1, ..., t_n)$. If $u \in O(t)$ then the *subterm* $t|u$ at a position u is defined as follows: $t|\lambda = t$ and $F(t_1, ..., t_n)|i \cdot u = t_i|u$. A subterm s of a term t is *proper* if $s \neq t|\lambda$. The *depth* of a subterm of t at position u is the length of u. Similarly *levels* of a term are defined. Given a term t, $level(t) = 0$ if t is a variable, and $level(F) = 1$ and $level(F(t_1, ..., t_n)) = 1 + \text{the maximum of } level(t_1), ..., level(t_n)$ if $t \equiv F(t_1, ..., t_n)$.

Contexts are terms in $Ter(\Sigma \cup \Box)$, in which the special constant $\Box$, denoting an empty place, occurs exactly once. Contexts are denoted by $C[\ ]$ and the result of substituting a term t in place of $\Box$ is $C[t] \in Ter(\Sigma)$. $s[v := t]$ means the substitution of a term t for the occurrence v in a term s.

A *substitution* is a map $\sigma : Var \to Ter(\Sigma)$ satisfying the equation $\sigma(F(t_1, ..., t_n)) = F(\sigma(t_1), ..., \sigma(t_n))$. The result l^σ of the application of the substitution σ to the term l is an *instance* of l. A *redex* (reducible expression) is an instance of a LHS of a rewrite rule. A reduction step $s \to t$ is a pair of terms of the form $C[l^\sigma] \to C[r^\sigma]$, where $l \to r$ is a rewrite rule in R. $\to^*$ denotes the reflexive transitive closure of $\to$.

A *normal form* is a term containing no redexes. A term t *has a normal form* n if t is reducible to n and n is anormal form.

The function symbol at λ of a term t is called the *principal function symbol* of t. Principal function symbols of LHSs of R are called *operators* and other function symbols *constructors*. A term t is an *operator term* (resp. a *constructor term*) if the leftmost symbol of t is an operator (resp. a constructor).

Let R be a TRS. A rewrite rule $l \to r$ is *left-linear* if no variable occurs twice or more in l.

R is *non-overlapping* if for any two LHSs s and t, any position u in t, and any substitution σ and τ : $Var \rightarrow Ter(\Sigma)$ it holds that if $(t|u)^{\sigma} = s^{\tau}$ then either $t|u$ is a variable or t and s are LHSs of the same rewrite rule and $u = \lambda$ (i.e. non-variable parts of different rewrite rules do not overlap and non-variable parts of the same rewrite rule overlap only entirely). R is *orthogonal* if its rules are left-linear and non-overlapping. Orthogonal systems are called *constructor systems* if the LHSs of rules do not include a proper operator term.

3. Strongly Sequential Systems

A redex in a term t is *needed* if it must be contracted in order to get to the normal form of t, and the *call-by-need* computation means that no redex is ever reduced unless it is needed. Neededness is not decidable in orthogonal TRSs, hence Huet & Lévy proposed a decidable approximation known as *strong sequentiality*[1]. However, strongly sequential systems are still too complicated to be used practically, and several simpler versions are proposed.

Definition 1 Let R be an orthogonal TRS and Ω be a new constant symbol. The set Ter_{Ω} of *Ω-terms* consists of signature $(\Sigma \cup \{\Omega\})$.

1. On $Ter_{\Omega}(R)$ we define a partial order $\leq$ by:
   - $\Omega \leq t$ for all $t \in Ter_{\Omega}(R)$.
   - $F(t_1, \ldots, t_n) \leq F(s_1, \ldots, s_n)$ if $t_i < s_i$ for $i = 1, \ldots, n$. We write $t < s$ if $t \neq s$ and $t \leq s$.
2. $s, t \in Ter_{\Omega}$ are said to be *compatible* (notation $s \uparrow t$) if there exists a term $p \in Ter_{\Omega}$ such that $s < p$ and $t < p$. Otherwise, s and t are *incompatible* (notation $s \# t$).
3. Let $S \subseteq Ter_{\Omega}$ and $t \in Ter_{\Omega}$. Then $t \geq S$ (respectively, $t \uparrow S$) if there exists some $p \in S$ such that $t \geq p$ (respectively, $t \uparrow p$); otherwise $t \not\geq S$ (respectively, $t \# S$).
4. t_{Ω} denotes the *Ω-term* obtained from a term t by replacing each variable in t with Ω. The *redex schemata* of a TRS R is $Lhs = \{l_{\Omega} \mid l \rightarrow r \in R\}$.
5. *ω-reduction*, written as $\rightarrow_{\omega}$, is defined on Ter_{Ω} as $C[t] \rightarrow_{\omega} C[\Omega]$ if $t \uparrow Lhs$ and $t \neq \Omega$. A redex w.r.t. $\rightarrow_{\omega}$ is an *ω-redex*.

Ω means 'unknown' or 'bottom'. Ordering relation $s < t$ is read "s is weaker than t" or "t is stronger than s".

Definition 2 Let R be a TRS and s and t Ω-terms of $Ter_{\Omega}(R)$.

1. A predicate P is *monotonic* when it is considered as a function mapping from an Ω-term to truth-values ordered by *false* < *true* such that if $s \leq t$, then $P(s) < P(t)$.
2. A monotonic predicate P is *sequential at a term* t if:$(\neg P(t) \wedge \exists s > t.\ P(s)) \Rightarrow \exists u.(t|u \equiv \Omega \wedge \forall s \geq t\ P(s) \Rightarrow s|u \neq \Omega)$
   An occurrence u as above is an *index* of P in t. P is *sequential* if it is sequential at every term which is in normal form (by the original rules of the system, not ω-reduction). $I_P(t)$ is the set of indexes of P in t.
3. $s \rightarrow' t$ if and only if $t \equiv s[u := q]$ for some redex occurrence u at s and an arbitrary term q. A predicate *nf'* of *strong normal form* is defined w.r.t. the reduction $\rightarrow'$ such that $nf'(s) = true$ if $s \rightarrow'^{*} t$ for some t in normal form.
4. An orthogonal system R is *strongly sequential* if *nf'* is a sequential predicate.

Strongly sequential systems are sequential, but not vice versa. Let $\Omega(t)$ be the result of substituting Ω for every outermost redex t. A *strongly needed redex* of t is a redex at an occurrence in $I_{nf'}(\Omega(t))$.

Proposition 3*[1] Indexes of strongly sequential systems are decomposable but not transitive.*

Definition 4 1. Let t be an ω-redex and $u \in I_{nf'}(s)$, $v_1, \ldots, v_n \in I_{nf'}(t)$. If $u \cdot v_i \in I_{nf'}(s[u := t])$ for every $s \in Ter_{\Omega}$ and $u \in I_{nf'}(s)$, then v_i is a *transitive index*.

2. A TRS R is called *index-transitive* if every ω-redex t has a transitive index.

Index-transitive at Definition 4 is the same as *transitive* at[7].

Proposition 5*[7] If a TRS R is index-transitive, then R is strongly sequential.*

Definition 6 1. Let $q, t \subseteq Ter_{\Omega}$. An occurrence u at t such that $t|u \equiv \Omega$ is said to be a *direction* for q if $t[u := \bullet] \# q$ ($\bullet$ is a 'fresh' symbol).

2. Let $Lhs^{*} = \{p \mid \Omega < p$ and p is a subterm of l for some $l \in Lhs\}$. A *transitive direction* is defined as a direction for Lhs^{*}.

Theorem 7 *[7] Let R be a TRS. R is index-transitive if and only if every simple ω-redex $t \in \{p \mid \Omega < p < r$ for some $r \in Lhs\}$ has a transitive direction.*

Strongly sequential constructor systems are orthogonal constructor systems which are strongly sequential. Strongly sequential constructor systems are index-transitive [8].

Often operators at proper subterms in the LHSs of rules make the system intricate while constructor systems are more manageable. We concern systems which can be transformed into strongly sequential constructor systems.

Definition 8 A simple ω-term is a closed ω-term s such that s is an operator term, $s \uparrow Lhs^{*}$, no proper subterm of s is an ω-redex, and s is not a redex.

Definition 9 A TRS R is *transformable* if every simple ω-term has a transitive direction.

Obviously transformable systems are strongly sequential, but not vice versa. For example, consider LHSs : $F(G(x, S(0)), B)$, $F(A, G(x, S(y)), G(1, 1)$. A system with these rules are strongly sequential, but it is not transformable as a simple ω-term $G(\Omega, S(\Omega))$ has no transitive direction.

4. Separable Systems

4.1 Separation trees

In the λ-calculus, the 'meaningless' of a term is explained by Böhm trees, which are considered to be the 'values' of terms. Together with Böhm trees, 'separability' is also well-developed[9]. The concept of meaningless-terms has been extended to orthogonal TRSs[10], in which there are several candidates for a notion of 'undefined' term on which to base the definition of Böhm trees. Common to these is the minimal condition that the proper subterms of left-hand sides should be meaningful. Based on this notion, we construct *separation trees*, which adapt to term rewrite systems the 'Böhm-out' construction of λ-calculus.

Definition 10 P_F is the set of tuples obtained by removing the principal function symbol F from the LHSs of F-rules.

1. An occurrence u is *useful* for P_F if $\forall p \in P_F$, $u \in O(p)$ and $p|u$ is not a variable. (For convenience, occurrences of P_F will be those of LHSs of the F-rules. So, every occurrence of P_F is not λ.)
2. A *separation tree* U for a set P_F is a tree, whose nodes are labelled by occurrences, such that
   - the root u_0 is a useful occurrence of P_F,
   - subtrees are separation trees of P_{Fi}, for $1 \le i \le n$, which do not re-use previous useful occurrences again, where P_F is partitioned into equivalence classes modulo the symbols at u_0 such that $P_F = P_{F1} \Sigma \cup \ldots \cup P_{Fn}$.
3. A separation tree U is *complete* if, in the result of recursive partition of P_F, the corresponding partitioned set for every leaf of U is a singleton; otherwise it is *partial*.
4. P_F is *distinct* if P_F has a complete separation tree. The F-rules are *separable* if P_F are distinct.
5. A constructor system is *separable* if every set of rewrite rules for an operator is separable.

Example 11

(i) $F(x, A, B, C) \rightarrow 1$
$F(B, x, A, C) \rightarrow 2$
$F(A, B, x, C) \rightarrow 3$

(ii) $F(x, A, B, C) \rightarrow 1$
$F(B, x, A, C) \rightarrow 2$
$F(A, B, x, D) \rightarrow 3$

(iii) $F(x, A, B, C, S(D)) \rightarrow 1$
$F(B, x, A, C, S(D)) \rightarrow 2$
$F(A, B, x, C, S(E)) \rightarrow 3$

The occurrence *4* is useful at (i). Because every symbol at *4* is the same, no partition is made, and then there is no other useful occurrence. Hence, (i) is not separable. (ii) and (iii) are separable.

The following shows that the selection order of useful occurrences is irrelevant to the success of constructing a separation tree.

Proposition 12 *Let the F-rules be separable and have more than one useful occurrences. Then choosing any one of them arbitrarily leads to a separation tree.*

Proof. Let P be a set of terms and u is a useful occurrence. Then, obviously, u is an useful occurrence of $P' \subset P$ as well. Given two useful occurrences u and v for P, u is 'optimal' if the number of partitioned subsets w.r.t. u is not less than the one w.r.t. v. Then, it is clear that there is at least one optimal useful occurrence for every set of terms. Partitioning into subsets as the order of u-v and v-u, we can get the same partitioned subsets. We claim that if the F-rules are separable, then there exists a complete separation tree U such that every node of U is optimal. If there are two or more than optimal occurrences at a certain stage, any successive application order by choosing them arbitrarily get the same result. Now we consider the case of choosing v when an optimal occurrence u and a non-optimal occurrence v at a certain stage. Then, obviously, at every subset after partitioning w.r.t. v, u is still a useful occurrence. Therefore, both choosing an optimal occurrence and non-optimal occurrence can leads to a complete separation tree. □

4.2 Transformable separation trees

Definition 13 A *transformable separation tree* U is a separation tree such that wherever there is a path from the root of U containing nodes labelled by u and v, and $u < v$, the node labelled by v is deeper than the node labelled by u.

Lemma 14 *Every separable system has a transformable separation tree.*

Proof. Let u be a useful occurrence with depth 2. Then, by the structure of terms, there exists another useful occurrence v such that $v < u$ and the depth of v is 2. By Proposition 12, there is a separation tree whose root is v. □

Lemma 15 *Suppose that the F-rules are in a constructor system. The F-rules are strongly sequential if and only if the F-rules are separable.*

Proof. Here, we present only a proof sketch. A detail proof can be found in[11]. Strongly sequential constructor systems are index-transitive, and then by Theorem 7 every simple ω-redex of the F-rules has a transitive direction. By Lemma 14 every separable system has a transformable separation tree. Then, the proof can be given by showing "every simple ω-redex of the F-rules has a transitive direction if and only if there is a complete transformable separation tree for the F-rules".

Consider a partitioning procedure to construct a complete transformable separation tree U. U is increasing as the partition proceeds. Then, we claim that there exists an equivalence class P partitioned by U and P has a useful occurrence u $\Leftrightarrow$ there exists a corresponding simple ω-redex s such that

$$u \text{ is a transitive direction of } s \text{ such that } s|u \equiv \Omega, \text{ and } s|v \not\equiv \Omega \Leftrightarrow v \in U,$$

where Ω means a subterm which has not been used previously for partitioning. We prove the above claim by using induction on the size of U. □

Theorem 16 *Let a system R be a constructor system. R is strongly sequential if and only if R is separable.*

5. Transformation

5.1 Generic transformation

In the implementation of TRSs, pattern-matching for some function symbols should be performed in advance of others. This notion should be preserved in transformation.

Definition 17 Let the F-rules be separable, O_u the set of some useful occurrences of P_F, $O_n = \{u \mid u$ is an every occurrence of $l \in P_F$ such that u is not useful for $P_F\}$, and $O_B = \{u \mid u$ is a disjoint-minimal occurrence in $O_n \cup O_u\}$. B's are subterms at O_B and A's are subterms other than B's in P_F. Given the F-rules and O_u, O_B is decidable and the *generic transformation* is applied to every F-rule as follows;

$$F(B_{11}, \ldots, B_{1n}, A_{11}, \ldots, A_{1m}) \rightarrow R_1$$
$$\ldots\ldots\ldots\ldots\ldots\ldots$$
$$F(B_{k1}, \ldots, B_{kn}, A_{k1}, \ldots, A_{km}) \rightarrow R_k$$

is transformed into

$$F(x_1, \ldots, x_n, A_{11}, \ldots, A_{1m}) \rightarrow F_1(x_1, \ldots, x_n, y_1, \ldots, y_h)$$

$$\cdots\cdots\cdots\cdots\cdots$$
$$F(x_1, \ldots, x_n, A_{p1}, \ldots, A_m) \rightarrow F_p(x_1, \ldots, x_n, y_1, \ldots, y_r)$$
$$F_1(B_{11}, \ldots, B_{1n}, y_1, \ldots, y_h) \rightarrow R_1$$
$$\cdots\cdots\cdots\cdots\cdots$$
$$F_i^j(B_{i1}, \ldots, B_{in}, y_1, \ldots, y_q) \rightarrow R_i$$
$$\cdots\cdots\cdots\cdots\cdots$$
$$F_i^j(B_{(i+j)1}, \ldots, B_{(i+j)n}, y_1, \ldots, y_q) \rightarrow R_{i+j}$$
$$\cdots\cdots\cdots\cdots\cdots$$
$$F_p(B_{p1}, \ldots, B_{pn}, y_1, \ldots, y_r) \rightarrow R_k$$

which satisfies the following conditions. (For convenience sake, the above rules describe the case that all B's are subterms at the level 2.)

1. If all symbols B's are variables in an F-rule, it remains unchanged. Otherwise, B's in the F-rules are replaced by new variables (not already occurring in the system being transformed) x's, the RHSs in the transformed F-rules consist of new operators and include no function symbols but all variables appearing in the LHS at arguments, where y's are variables occurring in A's. At this time, the replacement is made simultaneously for all B's. New rules with new operators $F_1, \ldots, F_p$ for $p \leq k$ are introduced.
2. Define $\simeq$ *equivalence relation* between two terms l and l' as follows:

   $l \simeq l' \Leftrightarrow$ *l and l' differ only the names of their variables.*

   The rewrite rules are partitioned, based on the equivalence classes of their transformed LHSs(i.e. the equivalence classes modulo symbols A's), and every such equivalence class is replaced by a single rule(every rewrite rule belongs to one of partitioned equivalence classes). In 1, if no equivalence class with two or more rules is created, then $p = k$; otherwise $p < k$ and, for an equivalence class with $j \geq 2$ rules in the original F-rules, there are corresponding j F_i-rules.
3. Let $P_i \subseteq R$ be an equivalence class created in the generic transformation and F_i-rules its corresponding new rules in the transformed system. For every $v \in O_B$ and every pair of LHSs l_1 and l_2 of P_i, there is a corresponding occurrence u and a pair of LHSs l_1' and l_2' of the F_i-rules such that u is at the level 2, $l_1|v \simeq l_1'|u$ and $l_2|v \simeq l_2'|u$.

In Definition 17, all function symbols appearing in the LHSs of the original system appear in the LHSs of a transformed system. All RHSs of the original F-rules never change and appear as RHSs in the transformed system. On the other hand, O_n is decidable. Hence, given the F-rules and O_u, O_B is decidable. The set O_A of occurrences of A's are useful and decidable such that $O_A = O_U - O_u$ for the set O_U of all useful occurrences of P_F.

Example 18 Suppose the following F-rules are given.

$$F(C(z, A), B, E) \rightarrow 1$$
$$F(C(B, z), A, E) \rightarrow 2$$
$$F(C(A, B), z, D) \rightarrow 3$$

Given $O_u = \{3\}$ and $O_u = \phi$, the rules are transformed as (i) and (ii), respectively. It is easy to see $O_n = \{1\cdot1, 1\cdot2, 2\}$. In (i), $O_B = \{1\cdot1, 1\cdot2, 2, 3\}$, and substitution of O_B creates a equivalence class with three rules, hence they are replaced with a single rule, and three new rules are created. In (ii), $O_B = \{1\cdot1, 1\cdot2, 2\}$, and substitution of O_B creates a equivalence class with two rules, hence they are replaced with a single rule, and two new rules are created.

(i) $F(C(x_1, x_2), x_3, x_4) \rightarrow F1(x_1, x_2, x_3, x_4)$
$F_1(z, A, B, E) \rightarrow 1$
$F_1(B, z, A, E) \rightarrow 2$
$F_1(A, B, z, D) \rightarrow 3$

(ii) $F(C(x_1, x_2), x_3, E) \rightarrow F_1(x_1, x_2, x_3)$
$F(C(x_1, x_2), x_3, D) \rightarrow F2(x_1, x_2, x_3)$
$F_1(z, A, B) \rightarrow 1$
$F_1(B, z, A) \rightarrow 2$
$F_2(A, B, z) \rightarrow 3$

The generic transformation can be described by transformable separation trees. Suppose a transformable separation tree U in a form of Fig. 1, where U_A consists of O_A in Definition 17. After the generic transformation, the pattern-matching sequence for U_A is performed in the transformed F-rules and the pattern-matching sequence for U_i's $(1 \leq i \leq p)$ in the new rules. By Definition 17.3, when some subterms in the patterns of an equivalence class created by the generic transformation are moved to

patterns of new rules, their 'structure' is preserved in the new rules. It is not difficult to see that, for every U_i, there is a corresponding separable trees V_i in the corresponding new rules so that there is a one-to-one corresponding mapping between U_i's and V_i's. By Proposition 12, we don't have to care what the specific structures of U_i's and V_i's are.

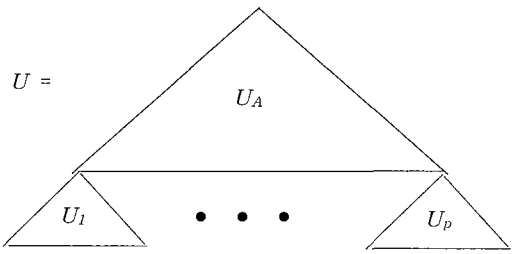


Fig. 1 Transformable separation tree

Lemma 19 *A set of rewrite rules is separable if and only if its transformed rules satisfying the generic transformation are separable.*

5.2 Transformation functions

In this subsection, we demonstrate how transformable systems are transformed. Three transformation functions *Co*, *T2*, and *ES* are introduced. The application of *Co*(resp. *T2* and *ES*) to a transformable system R is written as *Co(R)* (resp. *T2(R)* and *ES(R)*).

Definition 20 Let R be a separable system whose patterns are not deeper than the level 2 and $F(t_1, ..., t_n)$ be a LHS. We say that F *pattern-matches at the i-th place* if t_i is not a variable. If there is a useful occurrence i for the F-rules, then i is an *always-matched* place of F. If there is no function symbol at i for all F-rules, then i is a *never-matched* place of F. Otherwise, i is a *sometimes-matched* place of F. R is said to be *flat* if R has no sometimes-matched patterns.

5.2.1 Operator terms in patterns

In Example 21.(i), proper subterms *G(x, S(0))* and *G(x, E)* in LHSs have an operator G. The LHSs of these rules are transformed into terms in constructor form by adding new rules, whose corresponding LHSs are *G(x, S(0))* and *G(x, E)* and RHSs consists of new function symbols C_1 and C_2 and variables occurring at LHSs, and by replacing subterms of *G(x, S(0))* and *G(x, E)* in the original rules by RHSs of the new rules, which transforms the LHSs of the F-rules into $F(C_1(x), B)$ and $F(C_2(x), C)$.

In Example 21.(ii), an operator G is nested in the pattern. By one application of *Co*, a new rule $G(G(A)) \rightarrow C_1$ is created, which still has an operator G in the pattern. Another application of *Co* transforms them into a constructor system.

In Example 21.(iii), two proper subterms *G(1, x)* and *G(1, y)* of LHSs are $\simeq$ equivalent, $G(1, x) \simeq G(1, y)$. Such $\simeq$ equivalent operator terms are replaced by a same constructor term $C_1(x)$, and only one new rule whose LHS is *G(1, x)* is introduced.

Example 21

(i) $F(G(x, S(0)), B) \rightarrow 1$
$F(G(x, E), C) \rightarrow 2$
$G(D, D) \rightarrow 3$
is transformed into
$F(C_1(x), B) \rightarrow 1$
$F(C_2(x), C) \rightarrow 2$
$G(x, S(0)) \rightarrow C_1(x)$
$G(x, E) \rightarrow C_2(x)$
$G(D, D) \rightarrow 3$

(ii) $F(G(G(A))) \rightarrow 1$
$G(B) \rightarrow 2$
is transformed into
$F(C_1) \rightarrow 1$
$G(C_2) \rightarrow C_1$
$G(A) \rightarrow C_2$
$G(B) \rightarrow 2$
$G(A, B) \rightarrow 3$

(iii) $(G(1, x), A) \rightarrow 1$
$F(B, G(1, y)) \rightarrow 2$
$G(A, B) \rightarrow 3$
is transformed into
$F(C_1(x), A) \rightarrow 1$
$F(B, C_1(x)) \rightarrow 2$
$G(1, x) \rightarrow C_1(x)$

Algorithm 22 (Co) Let s be an outermost proper operator term in LHSs. Then, replace s by a

new term $C(x_1, ..., x_n)$, where C is a new constructor and the x_i's are variables occurring in s, and make a new rule $s \rightarrow C(x_1, ..., x_n)$. If there exists another proper operator term s' in LHSs such that $s \simeq s'$, then s' is also replaced by $C(x_1, ..., x_n)$. Repeat this procedure until there exist no operators at proper subterms in the patterns.

Co is not an instance of the generic transformation f Definition 17.

5.2.2 Patterns at deeper than level 2

The transformation function *T2* has already been introduced as *Transformation-1* at[5]. *T2* eliminates sometimes-matched patterns as well as patterns whose level is greater than 2. Then, repeating *T2* transforms a strongly sequential constructor system into a flat system.

In the Example 23.(i), the first rule has the pattern *0* at level 3, and the second rule has the symbol *S* at the level 3 and *0* at 4, and there are sometimes-matched pattern at occurrences *1* and *2*. The algorithm 'trims' the LHSs to level 2, by replacing deeper subterms and sometimes-matched patterns by new variables. Then, two new LHSs, $H(y, x, S(z))$ and $H(x, y, S(x))$, are obtained. Since $H(y, x, S(z)) \simeq H(x, y, S(z))$, these two rules are replaced by a single rule $H(x, y, S(z)) \rightarrow H_1(x, y, z)$ where H_1 is a new function symbol. This rule does the pattern-matching common to the first and the second rule, down to level 2 for always-matched patterns. By adding two rules, $H_1(C, y, 0) \rightarrow 1$ and $H_1(x, C, S(0)) \rightarrow 2$ the rest of pattern matching is done. The second rules has the pattern *0* at level 3. Applying the algorithm again to the second rule, the Example 23.(i) is obtained.

Example 23

(i) $H(C, x, S(0)) \rightarrow 1$
$H(x, C, S(S(0))) \rightarrow 2$
is transformed into
$H(x, y, S(z)) \rightarrow H_1(x, y, z)$
$H_1(C, y, 0) \rightarrow 1$
$H_1(x, C, S(w)) \rightarrow H_2(w)$
$H_2(0) \rightarrow 2$

(ii) $F(G(G(A))) \rightarrow 1$
$G(B) \rightarrow 2$
is transformed into
$F(G(x)) \rightarrow F1(x)$
$F1(G(A)) \rightarrow 1$
$G(B) \rightarrow 2$

The algorithm of *T2* can be defined simply by using the Definition 17; define the set O_u of useful occurrences whose levels are greater than 2.

5.2.3 Sometimes-matched patterns

A transformation function *ES* has already been introduced as the *Transformation-2* at[5]. In Example 24.(i), the occurrence *1* of the *F*-rules is sometimes-matched. Subterms of *Nil* and $Cons(x, y)$ are replaced by fresh variables, and rules of a new function symbol F_1 are added. In this replacement, an equivalent class is created. The second and third rule of Example 24.(i) become equivalent, so they are replaced by a single rule: $F(x_1, Cons(x, y)) \rightarrow F_1(x_1, x, y)$.

Rules in Example 24.(ii) are not strongly sequential. The G_1-rules are introduced by *ES*, and then they cannot be simplified further.

Example 24

(i) $F(x, Nil) \rightarrow 1$
$F(Nil, Cons(x, y)) \rightarrow 2$
$F(Cons(x, y), Cons(x, y)) \rightarrow 3$
is transformed into
$F(x, Nil) \rightarrow 1$
$F(x_1, Cons(x, y)) \rightarrow F_1(x_1, x, y)$
$F_1(Nil, x, y) \rightarrow 2$
$F_1(Cons(x, y), z) \rightarrow 3$

(ii) $G(x, A, B, C) \rightarrow 1$
$G(B, x, A, C) \rightarrow 2$
$G(A, B, x, C) \rightarrow 3$
is transformed into
$G(x, y, z, C) \rightarrow G1(x, y, z)$
$G1(x, A, B) \rightarrow 1$
$G1(B, y, A) \rightarrow 2$
$G1(A, B, x) \rightarrow 3$

The algorithm of *ES* is simply obtained by defining $O_u = \psi$ in the Definition 17.

5.3 Transformation of transformable systems

Lemma 25 *If R is transformable then Co(R) is a strongly sequential constructor system.*

Proof. Immediate from Definition 9. □

Theorem 26 *If R is transformable, then both $T2(Co(R))$ and $ES(Co(R))$ are flat.*

Proof. By Lemma 25 $Co(R)$ is strongly sequential (or separable). $T2$ and ES satisfy the generic transformation. Then, by Lemma 25 $T2(Co(R))$ and $ES(Co(R))$ are separable with whose LHSs have levels no more than 2. □

6. Correctness

In this section, transformation means the transformation of transformable systems by Co, $T2$, or ES.

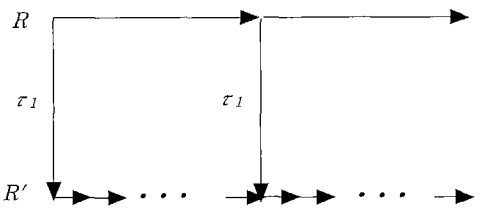
Fact 27 Let τ_I be one of transformation functions Co, $T2$, and ES, and suppose a transformable system R is transformed into a system R'; i.e. $\tau_I : R \rightarrow R'$. Then the following facts are observable.

1. R' consists of all signature of R and some new function symbols; $Ter(R') \supseteq Ter(R)$.
2. The reduction steps of R' are finer than R. For every one step reduction of R, there is at least one or more corresponding reduction steps in R': $s \rightarrow_R t \Rightarrow \tau_I(s) \rightarrow_{R'}^{+} \tau_I(t)$.
3. All normal forms of R are not normal form in R'. Hence, normal forms of R and R' are not the same. In Example 21.(ii), $G(A)$ is normal form in R but not in R'. In Example 23.(i), $H(C, 0, S(A))$ is normal form in R but not in R'.
4. If a term s has normal form in R, s has normal form in R'; the notion of normalizing is preserved.

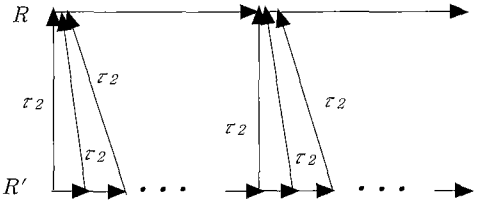
Definition 28 A *reduction graph* of a term s is rooted directed graph labeled as follows.

- Each node is labeled with a term.
- For each arc, the term labeling its source is reducible in one step to the term labeling its target.
- The root of the graph is labeled with s, and all nodes are accessible from the root.

Fig. 2.(i) shows that a reduction of R is implemented by multiple 'finer' reduction steps of R'. Fig. 2.(ii) means the same simulation of Fig. 2.(i) but it says one more thing that every 'finer' reduction step of R' is a part of implementation of a 'coarser' reduction step of R. If a transformation satisfies Fig. 2.(i) and Fig. 2.(ii), then R and R' *bisimulate*.



(i) Mapping R to R'



(ii) Mapping R' to R

Fig. 2 Bisimulation (R, R', τ_1, τ_2)

For a reduction $s \rightarrow_R t$, there are multiple-step reductions $\tau_1(s) \rightarrow_{R'}^{*} \tau_1(t)$ in R'. The last reduction step to reach to $\tau_1(t)$ is called an *external step*, and all other reduction steps starting from $\tau_I(s)$ are called *internal steps*. Terms of R' reached by internal steps are called *internal terms*. Terms of R' which are not internal are called *original terms*, which are the same as the terms of R. Note that there is a one-to-one correspondence between external steps of R' and reduction steps of R.

Fact 29 Suppose the system R is transformed into the system R'. Let $R'' \subseteq R'$ be a set of rules whose RHSs are newly created by transformation. Then, we can see that the newly created RHSs has only one function symbol at the root, and all its arguments are variables and linear if they exists. Consider a set S of rewrite rules obtained by swapping the LHSs and the RHSs in R''. Then, S has the following properties.

- S is strongly normalizing; in $T2$ and ES, the

principal function symbols of LHSs of S do not appear at RHSs, and, in Co, they appear at RHSs but not recursively.
- S is orthogonal.
- A term t is normal form in S if and only if $t \in Ter(R)$.
- Given an internal term of R', S returns a corresponding original term. Given an original term, S returns it.

We define a function $rev : Ter(R') \rightarrow Ter(R)$ as the set S of rewrite rules.

Definition 30 Suppose a system R is transformed into a system R' by a transformation function $\tau_1 : R \rightarrow R'$ and its reverse mapping is $\tau_2 : R' \rightarrow R$.

1. The mapping τ_1 is an *adequate forward mapping* if:
   - $\tau_1(s) \equiv s$ for every $s \in Ter(R)$.
   - If $s \in Ter(R)$ has a normal form R, then $\tau_1(s)$ has normal form in R'.
   - For every reduction $s \rightarrow_R t$, there exist reductions $\tau_1(s) \rightarrow_{R'}^* \tau_1(t)$.
2. The mapping τ_2 is an *adequate backward* mapping if:
   - $\tau_2 = rev$ (which is surjective).
   - If $s \in Ter(R')$ is normal form, $\tau_2(s)$ is normal form in R.
   - For $s,t \in R'$, if $s \rightarrow_{R'}^* t$, there exist reductions $\tau_2(s) \rightarrow_{R'}^* \tau_2(t)$.
3. Suppose the above 1 and 2 hold. Transformation (R, R', τ_1, τ_2) is *adequate bisimulation* if:
   - $\tau_1(\tau_2(t))$ is convertible to t and $\tau_2(\tau_1(s)) \equiv s$ for all $t \in Ter(R')$ and for all $s \in Ter(R)$.
   - $\tau_2(t) \rightarrow_R s \Leftrightarrow t \rightarrow_{R'}^* \tau_1(s)$ for $s \in Ter(R)$ and $t \in Ter(R')$.

Fact 27 and Fact 29 support the following simulation properties.

Lemma 31 *(Soundness of simulation). Suppose a system R is transformed into a system R' by one of transformation functions Co, T2, and ES. Then, there exists an adequate forward mapping $\tau_1 : R \rightarrow R'$.*

Lemma 32 *(Completeness of simulation) Suppose a system R is transformed into a system R' by one of Co, T2, and ES. Then, there is an adequate backward mapping $\tau_2 : R' \rightarrow R$.*

Theorem 33 *Suppose a transformable system T is transformed into a flat system F by Co, T2, and ES. Then, the transformation (T, F, τ_1, τ_2) is an adequate bisimulation.*

Proof. Suppose that T is transformed into F by one-step transformation. Then, by Lemma 31 and Lemma 32, there are adequate forward and backward mappings, and then it is immediate that the transformation is an adequate bisimulation. In case of multi-step transformations, given two adequate forward mappings $\tau_1^1 : R_1 \rightarrow R_2$ and $\tau_1^2 : R_2 \rightarrow R_3$, it is obvious to see $\tau_1^2 \cdot \tau_1^1$ is also an adequate forward mapping. Similarly the composition of two adequate backward mappings is an adequate backward mapping, since they are surjective. Then, it is immediate that the composition of multi-step adequate bisimulations is also an bisimulation. □

7. Conclusion

Transformation into a flat system is used in the implementation of TRSs and functional languages. In this paper, we introduce transformation techniques and show the correctness of transformation. Those techniques enhance previous work[5] and are associated with well-developed theories such as strong sequentiality[1] and separability[9]. This work could be seen as a foundation for the correct implementation of TRSs and functional languages.

Acknowledgement

This work was supported by grant No. 2000-1-30300-010-3 from the Basic Research Program of the Korea Science & Engineering Foundation.

References

[1] G. Huet and J.-J. Lévy. Computations in orthogonal rewrite systems I and II. In Lassez and Plotkin[12], pp. 394-443. (Originally appeared as[13].).

[2] J.W. Klop. *Combinatory Reduction Systems*, volume 127 of *Mathematical Centre Tracts*. CWI, Amsterdam, 1980. PhD Thesis.

[3] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In van Leeuwen[14], chapter 15.
[4] J.W. Klop. Term rewriting systems. In Abramsky et al. [15], pp. 1-116.
[5] J.R. Kennaway. Implementing term rewriting languages in Dactl. *Theoretical Computer Science*, 72 : 225-249, 1990.
[6] L. Maranget. Two techniques for compiling lazy pattern matching. Technical Report 2385, INRIA, 1994.
[7] Y. Toyama, S. Smetsers, M.C.J.D. van Eekelen, and M. J. Plasmeijer. The functional strategy and transitive term rewriting systems. In Sleep et al. [16], pp. 61-75.
[8] J.W. Klop and A. Middeldorp. Sequentiality in orthogonal term rewriting systems. *Journal of Symbolic Computation*, 12 : 161-195, 1991.
[9] H.P. Barendregt. *The Lambda Calculus, its Syntax and Semantics*. North-Holland, second edition, 1984.
[10] Z. Ariola, J.R. Kennaway, J.W. Klop, M.R. Sleep, and F.J. de Vries. Syntactic definitions of undefined: On defining the undefined. In *Theoretical Aspect of Computer Software*, Springer-Verlag, Lecture Notes in Computer Science 789, pp. 543-554, 1994.
[11] S. Byun. *The Simulation of Term Rewriting Systems by the Lambda Calculus*. PhD thesis, Univeristy of East Anglia, 1994.
[12] J.-L. Lassez and G.D. Plotkin, editors. *Computational Logic: Essay in Honor of Alan Robinson*. MIT Press, 1991.
[13] G. Huet and J.-J. Lévy. Call-by-need computations in non-ambiguous. Technical Report 359, INRIA, 1979.
[14] J. van Leeuwen, editor. *Handbook of Theoretical Computer Science*, volume B: Formal Method and Semantics. North-Holland, Amsterdam, 1990.
[15] S. Abramsky, D. Gabbay, and T. Maibaum, editors. *Handbook of Logic in Computer Science*, volume II. Oxford University Press, 1992.
[16] M.R. Sleep, M.J. Plasmeijer, and M.C.J.D. van Eekelen, editors. *Term Graph Rewriting Theory and Practice*. John Wiley & Sons, 1993.

변 석 우

1976년 ~ 1980년 숭실대학교 전자계산(학사). 1980년 ~ 1982년 숭실대학교 전자계산(석사). 1982년 ~ 1999년 ETRI 책임연구원. 1988년 ~ 1994년 영국 University of East Anglia 전산학(박사). 1998년 ~ 현재 경성대학교 컴퓨터과학과 조교수. 관심분야는 프로그래밍언어 의미론, rewriting system, 함수형 프로그래밍, 정형 시스템 등