분리 시스템의 람다 계산법으로의 변환
(Translation of Separable Systems into the Lambda Calculus)

변 석 우 †
(Sugwoo Byun)

요 약 본 연구에서는 패턴을 갖는 항 개서 체계(TRS, Term Rewriting Systems)의 룰을 람다 계산법으로 코딩하는 변환 방법을 제시한다. Böhm의 *분리성* 이론에 따라 *차별화된 룰 패턴*을 갖는 *분리 시스템*은 람다 계산법으로 변환될 수 있음을 보인다. 또한, *Böhm 동등 부류*의 특성을 적용함으로써, 이 변환은 디폴트 룰로 된 개서 시스템을 코딩할 수 있으며 TRS의 '의미 없는 텀'들을 동일한 람다 텀으로 해석할 수 있도록 한다.

키워드 : 분리성, 항 개서 시스템, Böhm 트리, 람다 계산법, 변환

Abstract This research presents an translation technique of encoding rewrite rules with patterns into the lambda calculus. We show, following the theory of *Böhm's separability*, rewrite rules with *distinctive patterns*, called *separable systems*, can be translated into the lambda calculus. Moreover, according to the property of *Böhm equivalence classes*, we can also encode rewrite systems with default rules, which allows to interpret some of 'undefined' terms of TRSs as an identified lambda term.

Key words : Separability, Term Rewriting Systems, Böhm trees, Lambda Calculus, Translation

1. Introduction

In this paper, we discuss the translation of TRSs (term rewriting systems) into the lambda calculus, which was raised as an open question [1]. An essential distinction between two systems is that TRSs use function symbols while the lambda calculus does not. We show pattern matching of rewrite systems can be represented by the lambda calculus.

In lambda-definability, it is already known that lambda-definable functions should be *sequential* so that non sequential functions such as $\{Por(x, T) \rightarrow T, Por(T, x) \rightarrow T, Por(F, F) \rightarrow F\}$ cannot be defined in the lambda calculus [2]. Another source of lambda-definability is *Böhm's separability*; given a set of *distinct* lambda terms $\{X_1, ..., X_n\}$, the lambda term F satisfying equations $FX_1 = Y_1 ... FX_n = Y_n$ for arbitrary lambda terms $Y_1, ..., Y_n$, is decided effectively [3]. Following the *Böhm's separability*, we define *separable systems*, a TRS version of separability, and show separable systems can be encoded into the lambda-calculus, following faithfully Böhm's separability.

This paper extends my previous work [4] by upgrading its exposition and adding the case of default rules, used widely in the functional programming with some strategy of rule searching. Default rules also are useful in identifying 'undefined' terms to encode them.

The paper is organized in the following way; Section 2 introduces Böhm trees, Böhm-out trans-

· This work was supported by Kyungsung University in 2005.

† 정 회 원 : 경성대학교 컴퓨터정보학부 교수
swbyun@ks.ac.kr
논문접수 : 2007년 12월 11일
심사완료 : 2008년 1월 10일

정보과학회논문지: 시스템 및 이론 제35권 제4호(2008.4)

formation, and separability in the lambda calculus, Section 3 defines separability in TRS following the separability of the lambda calculus, Section 4 presents encoding in detail together with examples including default rules and the correctness of translation, and finally Section 5 comments related works and the meaning of this work. We assume readers are familiar with the lambda calculus and the orthogonal term rewrite systems.

2. Böhm Trees and Böhm-Out Transformation

2.1 Böhm trees

In this section, we brief Böhm trees and Böhm-out transformation introduced at Section 10.3 and 10.4 of [5]. In the lambda calculus, the notion of unsolvability is based on the hnf (*head normal form*) rather than *normal form*. Given a lambda term M, $BT(M)$, called the *Böhm tree* of M, is defined inductively based on hnf; $BT(M) = \perp$ (null) if M has no hnf, and $BT(M) = \lambda x_1...x_k.yBT(M_1)...BT(M_n)$ if M is of the form $\lambda x_1...x_k.yM_1...M_n$. The set $O(BT(M))$ of *occurrences* (or *positions*) of $BT(M)$ is defined as follows; if $BT(M) = \perp$, $O(BT(M)) = \varepsilon$; $BT(M)$ has only one occurrence for the root. If $BT(M) = \lambda x_1...x_k.yBT(M_1)...BT(M_n)$, $O(BT(M)) = \{\varepsilon\} \cup \{i \cdot \alpha_1 \mid 1 \le i \le n \text{ and } \alpha_1 \in O(BT(M_i))\}$. $BT(M)_\alpha$, α subtree at an occurrence α of $BT(M)$, *is also defined inductively as follows;* $BT(M)_\alpha = BT(M)$ if $\alpha = \varepsilon$, and $BT(M)_\alpha = BT(M_i)_{\alpha_1}$ if $BT(M) = \lambda x_1...x_k.yBT(M_1)...BT(M_n)$. and $\alpha = i \cdot \alpha_1$.

Definition 1 (Böhm equivalence $\sim$)

(1) Let two lambda terms $M \equiv \lambda x_1...x_n.yM_1...M_m$ and $N \equiv \lambda z_1...z_{n'}.y'N_1...N_{m'}$ be in hnf. Then, M is equivalent to N, written $M \sim N$, iff $y \equiv y'$ and $n - m = n' - m'$.

(2) Let A and B be Böhm trees. Then $A \sim_\alpha B$ iff $A_\alpha \sim B_\alpha$.

(3) $M \sim_\alpha N$ iff $BT(M) \sim_\alpha BT(N)$.

(4) $A|\alpha$ denotes the label at the node α in $BT(A)$.

For example, $\lambda x.xM \sim \lambda yz.yMN$, but $\lambda x.yM \not\sim \lambda y.yM$.

Definition 2 (1) An occurrence a is *useful* for $\mathcal{T}$ if $\forall M \in \mathcal{T}$, $a \in O(BT(M))$ and M_a is not empty. and $\exists M,N \in \mathcal{T}$ $M \not\sim_a N$.

(2) $\mathcal{T}$ is *distinct* if $\mathcal{T}$ consists of one elements, or some occurrence a is *useful* for $\mathcal{T}$ and $\sim_a$ equivalence class of elements of $\mathcal{T}$ are all distinct.

(3) Two lambda terms M and N *agree up to* a if $\forall \beta < a$, $M|\beta = N|\beta$.

Church numerals, written as $\underline{n}$ for $n \in N$ including 0, have some good properties. Consider $\underline{0} \equiv \lambda fx.x$, $\underline{1} \equiv \lambda fx.x$, $\underline{2} \equiv \lambda fx.fx$, and $\underline{n} \equiv \lambda fx.f^n x$, which are normal forms. From the view of Böhm trees, we can observe that $\underline{0} \not\sim \underline{1} \sim \underline{2} \sim \underline{3} \ldots$. *Also* $\underline{1} \not\sim_1 \underline{2} \sim_1 \underline{3} \sim_1 \underline{4}$ .... and $\underline{2} \not\sim_{1\cdot 1} \underline{3} \sim_{1\cdot 1} \underline{4} \ldots$.

Lemma 3 For every $n \in N$ including 0, there exists a useful path $a = 1^n \equiv 1 \cdot 1 \cdot \cdots \cdot 1$, n-times repeated occurrences of 1, such that $\underline{n-1}$ and $\underline{n}$ agree up to a, $\underline{n} \not\sim_a \underline{n+1}$, and $\underline{n+1} \sim_a \underline{n+2} \sim_a \underline{n+3} \ldots$. Hence, every set of Church numerals is distinct.

Definition 4 Let $\mathcal{T} = \{M_1,..., M_p\}$ be a set of closed lambda terms. $\mathcal{T}$ is called *separable* if $\forall N_1,...,N_p \in \Lambda$ $\exists F \in \Lambda$ $FM_1 = N_1 \wedge \ldots \wedge FM_p = N_p$.

Theorem. 5 [5] In the lambda calculus, $\mathcal{T}$ *is distinct* $\Rightarrow$ $\mathcal{T}$ *is separable.*

2.2 Böhm-out transformation

In this subsection, we introduce a transformation function π, called *a-$\mathcal{T}$-faithful transformation*, following Definition 10.3.12 of [5]. Suppose an occurrence a is given and two lambda terms M and N agree up to a. Applying π to the $BT(M)$, written as M^π, has the following condition; $M \sim_a N$ *iff* $M^\pi \sim N^\pi$, and $M|a$ *is defined iff* M^π *solvable.* The main condition of π is that it should preserve the definedness/undefinedness of original terms.

Corrado Böhm showed that, given an set $\mathcal{T}$ of distinctive lambda terms, a lambda term F, satisfying equations at Definition 4, exists. Moreover, such F is a *a-$\mathcal{T}$-faithful transformation* and can be obtained by a constructive proof of Theorem 5. In this subsection, we define the constructive proof as an algorithm. Roughly F can be constructed as follows.

Suppose two lambda terms M and N are distinct, including subterms such that $BT(M)$ and $BT(N)$ agree up to α and $M \not\sim_a N$. The transformation π takes out subterms at α of $BT(M)$ and $BT(N)$

and let them be M^*_α and N^*_α, respectively. M^*_α and N^*_α are substitution instances of M_α and N_α, respectively. π preserves both $\sim_\alpha$ and $\nsim_\alpha$ equivalence classes. As $M \nsim_\alpha N$, $M^*_\alpha \nsim N^*_\alpha$ holds. Then, to complete equations of Definition 4, the next phase of π is to decide appropriate RHS (right-hand sides) of equations.

There are two main elementary functions of $U^n_i \equiv \lambda x_1 \ldots x_n . x_i$, called *selection* and $P_n \equiv \lambda x_1 \ldots x_n . \langle x_1 \ldots x_n \rangle = \lambda x_1 \ldots x_n x_{n+1} . x_{n+1} x_1 \ldots x_n$, called *permutation*.

By π_α defined at Algorithm 6, given α and $\mathcal{T}$, we find out a context $(\ )^{\pi\alpha} = [\ ]\ L_1 \ldots L_n$ such that $M^{\pi\alpha} = BT(M)^*_\alpha$. π_α is a finite compositions of three basic transformations π_f, π_o, and π_s.

Algorithm 6 (Böhm-out transformation).

Let $\mathcal{T}$ be a set of lambda terms and $\forall M \in \mathcal{T}$ $\alpha \in O(BT(M))$. The Böhm out transformation π_α is a (repeated) sequence of π_f, π_o, and π_s defined as follows.

(1) π_f - transform $\mathcal{T}$ into *λ-free form*, having no λ -abstraction at the root.

$$(\ )^{\pi f} = (\ )\ x_1 \cdot \cdot \cdot x_n$$

where $x_1 \ldots x_n$ are new variables and $n = max(n_i \mid n_i$ *is the number of head abstractions of* $M_i \in hnf(\mathcal{T}))$.

(2) π_o - transform $\mathcal{T}^{\pi f}$ into *original form*; if $M \equiv \vec{\lambda x}.y\vec{M}$, then $y \notin FV(\vec{M})$.

$$(\ )^{\pi o} = (\ )\ a_1 \cdot \cdot \cdot a_{p+1},\ [y := P_p],$$

where y is the head variable of an $\sim$ equivalence class in $\mathcal{T}^{\pi f}$, p is the maximum number of success terms following y at $\mathcal{T}^{\pi f}$, and $a_1 \cdot \cdot \cdot a_{p+1}$ are new variables. After performing π_o, every $\sim$ equivalence class has a distinct head variable a_i $(1 \le i \le p+1)$.

(3) π_s - select one of following success terms.

$$(\ )^{\pi s} = (\ )\ [z := U^n_j],$$

where z is a head variable, n is the number of terms following z, and j is the prefix of α such that $\alpha = j \cdot \alpha'$.

(4) Repeat the above procedures with $(((\mathcal{T})^{\pi f})^{\pi o})^{\pi s}$ until α' becomes empty.

π_α is the composition of sequences of π_f, π_o:

$$(\ )^{\pi\alpha} = [\ ]\ x_1 \ldots x_n\ a_1 \cdot \cdot \cdot a_{p+1}\ [y := P_p]\ [z := U^n_j] \cdot \cdot \cdot .$$

All transformations terminate in finitely many steps since α is finite.

3. Separability in TRS

We assume familiarity with orthogonal TRSs [6,7]. Principal function symbols of LHS (left-hand sides of rules) are called *operators* and other function symbols *constructors*. A term t is an *operator term* if the leftmost symbol of t is an operator. A constructor term is a term having no operators and variables. Orthogonal systems are called *constructor systems* if LHS do not include a proper operator subterm. If the principal function symbol of l is F in a rewriting rule $l \to r$, this rule is called an F-rule.

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: $O(t) = \{\varepsilon\}$ if t is a variable, and $O(t) = \varepsilon \cup \{i \cdot u \mid 1 \le i \le n$ *and* $u \in O(t_i)\}$, if t is of the form $F(t_1, \ldots, t_n)$. If u is an occurrence of a term t, $t|u$ denotes the subterm of t at u.

Separable systems, their properties, and their transformation have been introduced in my previous work [10]. Separable systems, a class of Orthogonal TRSs, can be transformed into a very simple systems, called a *flat system*, a constructor system having at most one constructor in LHS of the rules.

Theorem 7 [10] Let $\mathcal{R}$ be a TRS. $\mathcal{R}$ is separable if and only if there exists a flattening of transformation that $\mathcal{R}$ transformed is also separable.

Example 8 [10] Consider the following separable systems.

$H(G(A, A, x), A) \to t_0$
$H(G(A, x, A), B) \to t_1$
$G(B, B, B) \to t_2$

These rules can be transformed by *flattening* as follows

$H(G_A(x, y), z) \to H_{AGA}(x, y, z)$
$H_{AGA}(x, y, A) \to H_{GA}(x, y)$
$H_{AGA}(x, y, B) \to H_{GB}(x, y)$
$H_{GA}(A, x) \to t_0$
$H_{GB}(x, A) \to t_1$
$G_B(B, x) \to G_{BB}(x)$
$G_{BB}(B) \to t_2$
$G(A, x, y) \to G_A(x, y)$

$G(B, x, y) \rightarrow G_B(x, y)$.

The original H-rules at Example 8 is not a constructor system as they have an operator G in LHS while their transformed rules are constructor systems - G_A is a fresh constructor introduced in transformation. Flat systems can be encoded directly into the lambda calculus, and then separable systems also can be via flattening.

Separability implies strong sequentiality, but not vice versa. Also, Strong sequentiality with index transitivity implies separability, but not vice versa;

Strong Sequentiality $\supsetneq$ *Separability* $\supsetneq$ *Strong Sequentiality with Index Transitivity*

If we are confined to constructor systems, all those three are the same. In constructor systems, the notion of distinction is directly related to that of the lambda calculus. According to [8], at least the proper subterms of LHS should be meaningful. Based on this notion, we construct *separation trees* in orthogonal TRSs.

Definition 9 Let P_F be LHS of F-rules.

(1) An occurrence u which is not ε is *useful* for P_F if $\forall p \in P_F$, $u \in O(p)$ and $p|u$ is not a variable.

(2) A *separation tree* U_T for a set P_F is a tree, whose nodes are labeled by occurrences, such that
- the root u_0 is a useful occurrence of P_F,
- subtrees are separation trees of P_{Fi}, for $1 \le i \le n$, by not reusing previously useful occurrences again, where P_F is partitioned into equivalence classes modulo the symbols at u_0 such that $P_F = P_{F1} \cup \ldots \cup P_{Fn}$.

(3) A separation tree U_T is *complete* if, in the result of recursive partition of P_F, the corresponding partitioned set for every leaf of U_T is 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. A constructor system is *separable* if every set of rewrite rules for an operator is separable.

Example 10 (1) $P_F = \{(x, A, B), (B, x, A), (A, B, x)\}$ is not distinct.

(2) $P_F = \{(x, A, B, C), (B, x, A, C), (A, B, x, D)\}$ is distinct.

In Example 10.(2), the symbols at *4* separate P_F into two groups of $\{(x, A, B, C), (B, x, A, C)\}$ *and* $\{(A, B, x, D)\}$, and then, based on the symbols at *3*, $\{(x, A, B, C), (B, x, A, C)\}$ also is separated into $\{(x, A, B, C)\}$ and $\{(B, x, A, C)\}$, which are singletons.

A separation tree is related to a reduction strategy. For example, in a rewrite rule $F(A, B, x) \rightarrow 1$ and a term $F(Redex1, Redex2, Redex3)$, a needed reduction strategy may reduce either *Redex1* or *Redex2*, or reduce them simultaneously. However, if we follow reduction with a separation tree having an occurrence 2 is at the root, *Redex2* is reduced first. In this way. a separation tree chooses one of possible reduction paths. Like the Example 8, the same effect is obtained when flattening is applied.

4. Interpretation and Translation

4.1 Conditions of translation

There can be much variation in translating one system into another, over the basic notions to something in technical details. We sketch the conditions that our translation $\phi : Ter(\Sigma) \rightarrow \Lambda$ needs to hold.

C-1. (Preserving equality and inequality) The most basic condition is to preserve equality such that $t = t'$ *implies* $\phi(t) = \phi(t')$. However, this is not enough, since there can be a trivial translation such that, for example, $\forall t \in Ter(\Sigma)$, $\phi(t) = \mathcal{Q}$. Hence, a condition of some degree of inequality is needed; $t \neq t'$ *implies* $\phi(t) \neq \phi(t')$. This condition will hold in *constructor terms*, terms consisting of constructors.

C-2. (Homogeneous mapping) ϕ is a homogeneous function, a structure-preserving mapping; $\phi(F(M_1, \ldots, M_n)) = \phi(F)(\phi(M_1), \ldots, \phi(M_n))$.

C-3. (Mapping constructors to numerals) The lambda calculus doesn't use function symbols while TRSs do. In TRSs, constructors denote certain 'values' [8], on which distinction and equivalence of terms are based. Various interpretations of constructors into the lambda calculus are possible as long as they can preserve equality and inequality consistently. For example, constructors 0 and 1 in TRS can be translated $\underline{1}$ and $\underline{0}$, respectively. In our translation, constructors are translated into a

Church numerals as they holds good properties as shown in Lemma 3.

C-4. (Encoding operators based on Böhm-out transformation) Translation ϕ is defined following Böhm-out transformation. Like Definition 4, given a set of distinctive lambda terms obtained by encoding constructors, an encoded lambda terms for an operator is decided by Böhm-out transformation, described in Algorithm 6. Here. translation ϕ is mainly determined by LHS including distinctive patterns and operators, and information about RHS is used as little as possible.

4.2 Encoding terms

The set Σ of symbols of a TRS consists of a set Σ_c of constructors, a set Σ_x of variables, and a set Σ_f of operators; $\Sigma = \Sigma_c \cup \Sigma_x \cup \Sigma_f$. P_F is LHS of F-rules, and P is the set of P_G for all operators $G \in \Sigma_f$.

The function ϕ is specified by $\phi_c : \Sigma_c \rightarrow \Lambda^0$, $\phi_x : \Sigma_x \rightarrow \Lambda^0 \cup \square$, $\phi_f : \Sigma_f \rightarrow \Lambda^0$, and

$\phi_p : P \rightarrow \Lambda^0$. Given a term $T(a_1, ..., a_n)$, its encoding is defined;

$\phi(T(a_1, ..., a_n)) = \phi_c(T)\ \phi(a_1) \cdots \phi(a_n)$ if T is a constructor,

$= \phi_f(T)\ \phi_p(a_1, ..., a_n)$ if T is an operator.

Our encoding consists of two phases. First, functions ϕ_c, ϕ_x, and ϕ_p are defined such that ϕ_p satisfies the condition "if P is distinct, $\phi_p(P)$ is distinct". Then, ϕ_f is decidable by the construction of Böhm-out transformation. There are several ways to define ϕ_c, ϕ_x, and ϕ_p. Our encoding is based on Church numerals.

Definition 11 Suppose a constructor c with an arity k and a tuple of arguments for an operator $(a_1, ..., a_m)$. Every element of Σ_c is mapped uniquely to a natural number $n \in N$. Let a be the occurrence of a variable x in a given term.

- $\phi_c(c) = \lambda x_1 \cdots x_k.\langle \underline{n}, x_1 \cdots x_k \rangle$.
- $\phi_p(a_1, ..., a_m) = \langle \phi(a_1), ..., \phi(a_m) \rangle$
- $\phi_x(x) = \square$, if x is in a LHS
  $= (\ )^{\pi a}$, if x is in a RHS,

where a is determined by the occurrence of the corresponding x in LHS.

Special attentions are given to the encoding of variables. Variables occurring in LHS and RHS have different operational meaning. From the view of separability, a variable in a LHS cannot be a useful occurrence. Therefore, a special symbol $\square$ is introduced in our encoding to denote that an occurrence of $\square$ cannot be a useful occurrence in the lambda calculus. A variable in a RHS plays a role of a 'place-holder' for the corresponding variable in LHS. Instantiation of variables, determined at pattern matching, is applied to all corresponding place-holders in RHS. Our encoding represents this behavior.

Definition 12 Let $\mathcal{T}$ be a set of lambda terms in which every element of $\mathcal{T}$ includes at least one Church numeral. Then, a *numeral separation tree* U_λ, whose nodes are labeled by occurrences of elements of $BT(\mathcal{T})$, is defined as follows.

- the root of U_λ is a_0 such that, $\forall M \in \mathcal{T}$, M_{a0} is a Church numeral. If $\mathcal{T}$ is singleton, then $\mathcal{T}$ has a numeral separation tree. Otherwise, let $\mathcal{T}$ be partitioned into equivalence classes modulo Church numerals at a_0;
  $\mathcal{T} = \mathcal{T}_1 \cup ... \cup \mathcal{T}_n$.
- A subtree of a_0 is a numeral separation tree of $\mathcal{T}_i$, $1 \le i \le n$.
- A numeral separation tree is *complete* if, in the result partition of $\mathcal{T}$, the corresponding partitioned set for every leaf of U_λ is singleton.

In the above definition, a numeral occurrence itself is not useful. Hence, a numeral separation tree is not a separation tree. However, given a complete numeral separation tree, a separation tree can be obtained.

Lemma 13 If the set $\mathcal{T}$ of lambda terms including Church numerals has a complete numeral separation tree, then $\mathcal{T}$ is distinct.

Lemma 14 Let $P_F = \{P_1, ..., P_n\}$ be the set of patterns of F-rules in a separable system, U_T a separation tree of P_F, and $\mathcal{T} = \phi_p(P_F)$.

(1) Every element of $\mathcal{T}$ has a normal form.

(2) Let $u = u_1 \cdots u_m$ be a constructor occurrence at $P_i \in P_F$. Then, there exists a numeral occurrence a_1 in the normal form of $\phi_p(P_i)$ such that $a_1 = u_1 \cdot (u_2+1) \cdots (u_m+1) \cdot 1$.

(3) For every U_T, there exists a corresponding numeral separation tree U_λ in $\mathcal{T}$.

(4) $\mathcal{T}$ is distinct.

Example 15 Suppose a set of rewrite rules

$F(C(0), 1) \rightarrow 1$ and $F(C(1), 2) \rightarrow 2$, where C, 0, 1 and 2 are constructors, and let $P_1 = (C(0), 1)$ and $P_2 = (C(1), 2)$. Assume that $\phi_c(C) = \lambda x.x\langle\underline{3}, x\rangle$, $\phi_c(1) = \langle\underline{1}\rangle$, and $\phi_c(2) = \langle\underline{2}\rangle$. Then $\phi_p(P1) \equiv \langle\langle\underline{3}, \langle\underline{0}\rangle\rangle, \langle\underline{1}\rangle\rangle$ and $\phi_p(P2) \equiv \langle\langle\underline{3}, \langle\underline{1}\rangle\rangle, \langle\underline{2}\rangle\rangle$. For occurrences 1, $1 \cdot 1$, 2 of P_1 and P_2 of F-rules, there are corresponding numeral occurrences $1 \cdot 1$, $1 \cdot 2 \cdot 1$, $2 \cdot 1$ at $\phi_p(P_1)$ and $\phi_p(P_2)$.

Theorem 16 A separable system is directly translated into the lambda calculus.

Proof. By Lemma 14 and Theorem 5.

4.3 Example and correctness

Following Algorithm 6, Definition 11, and Lemma 14, we define an encoding function $\phi: \Sigma \rightarrow \Lambda^0$ (closed lambda terms). The following example is borrowed from [9].

Example 17 Consider the following simple rules.

$$F(0) = 1$$
$$F(2) = 3$$

Let $\phi_c(0) = \langle\underline{0}\rangle$ and $\phi_c(2) = \langle\underline{2}\rangle$. Then, $a = 1$.

$(1)\ (\ )^{\pi_1} = (\ )\ z$

$(\langle\underline{0}\rangle)^{\pi_1} = (\lambda z.z\ \underline{0})\ z = z\ \underline{0}$

$(\langle\underline{2}\rangle)^{\pi_1} = (\lambda z.z\ \underline{2})\ z = z\ \underline{2}$

$\{z(\lambda fx.x), z(\lambda fx.f(fx))\}$ is original.

$(2)\ (\ )^{\pi_S} = (\ )\ [z := \boldsymbol{U}^1_1]$ (the first term is selected according to the first prefix of $a = 1$)

$(z\ \underline{0})^{\pi_S} = \boldsymbol{U}^1_1\ \underline{0} = \underline{0} \equiv \lambda fx.x$

$(z\ \underline{2})^{\pi_S} = \boldsymbol{U}^1_1\ \underline{2} = \underline{2} \equiv \lambda fx.f(fx)$

$(3)\ (\ )^{\pi_1} = (\ )\ f\ x$

$(\lambda fx.x)^{\pi_1} = (\lambda fx.x)\ f\ x = x$

$(\lambda fx.f(fx))^{\pi_1} = (\lambda fx.f(fx))\ f\ x = f\ (f\ x)$

In $\{x, f(fx)\}$, every $\sim$ equivalence class has a single term, hence the transformation

π_a terminates. Now we apply π_1 transformation to encode RHS.

(4) $(\ )^{\pi 1} = (\ )\ [x := \phi(1)]\ [f := \lambda y.\ \phi(3)]$ (since f has one following term).

$(\ )^{\pi} = z\ f\ x\ [z := \boldsymbol{U}^1_1]\ [x := \phi(1)]\ [f := \lambda y.\phi$

$(3) = \boldsymbol{U}^1_1\ \lambda y.\phi(3)\phi(1)$

Then $\phi(F) \equiv \lambda x.x\boldsymbol{U}^1_1\ \lambda y.\phi(3)\phi(1)$.

We confirm the encoding ϕ.

$\phi(F(0)) = \phi(F)\langle\underline{0}\rangle$

$\equiv (\lambda x.x\boldsymbol{U}^1_1\ \lambda y.\phi(3)\phi(1))\langle\underline{0}\rangle$

$= \langle\underline{0}\rangle\boldsymbol{U}^1_1\ \lambda y.\phi(3)\phi(1)$

$= (\lambda z.z\underline{0})\boldsymbol{U}^1_1\ \lambda y.\phi(3)\phi(1)$

$= \boldsymbol{U}^1_1\ \underline{0}\ \lambda y.\phi(3)\ \phi(1)$

$= \underline{0}\ \lambda y.\phi(3)\ \phi(1) \equiv (\lambda fx.x)\lambda y.\phi(3)\phi(1)$

$= \phi(1)$,

which shows encoding $F(0) = 1$ in TRS.

F-rules of Example 17 are flat. Rewrite rules with more complex patterns can be also translated into the lambda calculus either via flattening as described at Theorem 7 or by following a path of a separation tree. The case of recursively-defined rules is introduced in the Appendix, where the $\boldsymbol{Y}$ combinator is used for the recursion.

Theorem 18 (Correctness) Let s and t be terms of a separable system and those have no *operator normal forms*, normal form including an operator.

(1) $s \rightarrow^* t \Rightarrow \phi(s) \rightarrow^*_\beta \phi(t)$.

(2) $\phi(s) \rightarrow^*_\beta \phi(t) \Rightarrow s \rightarrow^* t$.

(3) If s has a hnf (head normal form), then $\phi(s)$ has a β-hnf.

The condition of "operator normal terms" at Theorem 18 cannot be lifted, since they are β-reduced to strange terms in the translation according to the property of $\sim_a$-equivalence. Following the well-defined notion of hnf in the lambda calculus, we apply this notion to TRS; a hnf in TRS is a term having no redex at the root, also known as *root stable* in [8]. Because a nf (normal form) is a special case of hnf, Theorem 18.(3) implies "if s has a nf, then $\phi(s)$ has a β-nf". Theorem 18.(1) and (2) means that translation ϕ holds equality and inequality conditions. Hence, ϕ satisfies translation conditions C-1 to C-4.

4.4 Encoding default rules

Translation ϕ malfunctions for operator normal forms as we discussed in the above. Like usual functional programming languages, those can be interpreted as $\perp$, a fresh symbol denoting an 'undefined' value - this could be replaced by $\mathcal{L}$ in the pure lambda calculus. In this subsection, we discuss how to encode them.

At Example 17, 0 is interpreted as $\underline{0}$, and 2 as $\underline{2}$. According to the $\sim_a$-equivalence class of Lemma 3, $\phi(F(1)) = \phi(F(2)) = \phi(3)$ also holds. As we discussed in the translation condition C-3, interpreting constructors, we can have many alternatives as long as equality and inequality are preserved. A more refined way is that every constructor appearing in LHS is associated with ordinal numbers

starting from 0; $\phi_c(c_i) = \underline{i}$, for $I \in N$. Then, the next ordinals are assigned to constructors appearing only in the RHS. For example, constructors in Example 17 are mapped: 0 to $\underline{0}$, 2 to $\underline{1}$, 1 to $\underline{2}$, and 3 to $\underline{4}$.

According to the property of $\sim_\alpha$-equivalence and new ϕ_c, by adding default rules, we can extend translation ϕ such that all operator normal forms are identified to be undefined. For example, a default rule is added to Example 17.

Example 19 Addition of a default rule to rules at Example 17.

$F(0) = 1$
$F(2) = 3$
$otherwise = error$

Then, translation is applied as follows:

$\phi(F)(\phi(0)) = \phi(3)$
$\phi(F)(\phi(1)) = \phi(4)$
$\phi(F)(\phi(2)) = \phi(\perp)$

In this way, we can identify all operator normal forms in the translation. As new numbers are assigned for default rules, the notion of orthogonality is preserved, Hence, the correctness of Theorem 18 still holds. The meaning of Theorem 18 is also extended; translation preserves not only equalilty and inequality based on constructor terms but also a certain level of definedness/undefinedness of terms. This may lead us to more abstract discussion such as Böhm trees of TRSs [8] and translation preserving them, which is beyond this work.

In a practical aspect, default rules are widely used in functional programming with the strategy of 'top-to-bottom' pattern-matching.

Example 20 Consider following Factorial functions written in Haskell.

```
fac 0 = 1
fac n = n * fac (n-1)
```

These rules also can be translated by ϕ. The second rule of fac, a default rule, matches every numerals other than *0*. Encoding *0* into $\underline{0}$ and *n* (default case) into $\underline{1}$, one can encode the fac rules.

5. Related Works and Conclusion

In Berarducci and Böhm's *canonical systems* [11], for every pair of constructor and operator, one rule is defined, where exactly one constructor always appears at a fixed occurrence *1* in LHS, so no operator normal terms exist. Their encoding is so elegant that the $\boldsymbol{Y}$ combinator is not used for recursively-defined rules. However, it would be impossible for their technique to encode rewrite rules with less restricted patterns.

In this work, we show pattern-matching semantics of TRSs can be represented in the lambda calculus, following Böhm's separability. $\sim_\alpha$ equivalence of Böhm tree allows us to encode default rules and undefined terms of TRSs. We conjecture that translation ϕ would hold properties of separable systems more tightly than as described in this paper; neededness, SN, Böhm-trees of separable systems, and etc would be also preserved and reflected. We remain this issue as a future study.

This work can serve as a formalism of translating TRS, including functional programming languages, into the lambda calculus, since it follows faithfully Böhm's separability, a general theory applicable to a large class of equations.

References

[1] N. Dershowitz and J.-P. Jouannaud and J.W. Klop. Open problems in rewriting - no. 1. In *4th International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science 488, pp. 445-456, Springer-Verlag, 1991.

[2] G. Berry. Stable models of typed λ-calculi. In *Automata and Languages and Programming*. Lecture Notes in Computer Science 62, pp. 72-89, Springer-Verlag, 1978.

[3] C. Böhm. Alcune proprietà delle forme β-η -normalinel λ-K-calcolo. *IAC Pubbl.*, 696:19, 1968.

[4] S. Byun, J.R. Kennaway, and R. Sleep. Lambda-definable Term Rewriting Systems. In *Asian Computer Science Conference '96*, Lecture Notes in Computer Science 1023, pp 106-115, Springer-Verlag, 1996.

[5] H. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. North-Holland, 1984.

[6] J.W. Klop. Term rewriting systems. In Abramsky et al., editors, *Handbook of Logic in Computer Science*, volume II. Oxford University Press, 1992.

[7] G. Huet and J.-J. Lèvy. Computations in Orthogonal Rewrite Systems I and II. In Lassez and Plotkin, eds., *Computational Logic: Essay in Honor of Alan Robinson*, MIT Press 1991. (Originally appeared as Technical Report 359, INRIA, 1979.)

[8] J.R. Kennaway, F.J. de Vries and V. van Oostrom, Meaningless terms in rewriting. *Journal of Functional and Logic Programming*, 1999.
[9] S. Byun, Translation of Functions into the Lambda-Calculus. *기초과학연구소 논문집*, 제 15권, 3호, pp 261-269, 경성대학교 기초과학연구원, 2004.
[10] 변석우, 분리가능 시스템의 지수 추이성과 변환. *정보과학회논문지* 제31권 제5호, pp. 658-666, 2004년 5월.
[11] A. Berarducci and C. Böhm, A self-interpreter of lambda calculus having a normal form. In *CSL-92*, Lecture Notes in Computer Science 702, pp. 85-99, Springer-Verlag, 1992.

Appendix

Example.

Consider Peano's arithmetics, having recursively-defined rules, where $U_T = 1$.

$$A(0, x) \rightarrow x$$
$$A(S(x), y) \rightarrow S(A(x, y))$$

Let $\phi_c(0) = \langle\underline{0}\rangle$ and $\phi_c(S) = \lambda x.\langle\underline{2},x\rangle$. Then, $\phi_p(0, y) = \langle\langle\underline{0}\rangle, \square\rangle$, $\phi_p(S(x), y) = \langle\langle\underline{2}, \square\rangle, \square\rangle$, and $\alpha = 1 \cdot 1$.

(1) $(\ )^{\pi_1} = (\ )\ z_1$

$(\langle\langle\underline{0}\rangle, \square\rangle)^{\pi_1} = (\lambda z.z\langle\underline{0}\rangle\square)z_1 = z_1\ \langle\underline{0}\rangle\ \square$

$(\langle\langle\underline{2}, \square\rangle, \square\rangle)^{\pi_1} = (\lambda z.z\ \langle\underline{2}, \square\rangle\ \square)\ z_1 = z_1\ \langle\underline{2}, \square\rangle\ \square$

(2) $(\ )^{\pi_S} = (\ )\ [z_1 := \boldsymbol{U}^2_1]$ *(due to* $\alpha = 1 \cdot 1$*)*

$(z_1\ \langle\underline{0}\rangle\ \square)^{\pi_S} = \boldsymbol{U}^2_1\ \langle\underline{0}\rangle\ \square = \langle\underline{0}\rangle$

$(z_1\ \langle\underline{2}, \square\rangle\ \square\ )^{\pi_S} = \boldsymbol{U}^2_1\ \langle\underline{2}, \square\rangle\ \square = \langle\underline{2}, \square\rangle$

(3) $(\ )^{\pi_1} = (\ )\ z_2$

$(\langle\underline{0}\rangle)^{\pi_1} = (\lambda z.z\ \underline{0})\ z_2 = z_2\ \underline{0}$

$(\langle\underline{2}, \square\rangle)^{\pi_1} = (\lambda z.z\ \underline{2}\ \square)\ z_2 = z_2\ \underline{2}\ \square$

(4) $(\ )^{\pi_0} = (\ )\ a_1\ a_2\ [z_2 := \boldsymbol{P}_2]$

$(z_2\ \underline{0})^{\pi_0} = \boldsymbol{P}_2\ \underline{0}\ a_1\ a_2 = a_2\ \underline{0}\ a_1$

$(z_2\ \underline{2}\ \square)^{\pi_0} = \boldsymbol{P}_2\ \underline{2}\ \square\ a_1\ a_2 = a_1\ \underline{2}\ \square\ a_2$

(5) $(\ )^{\pi_S} = (\ )\ [a_2 := \boldsymbol{U}^2_1]\ [a_1 := \boldsymbol{U}^3_1]$ (due to the second prefix of a)

$(a_2\ \underline{0}\ a_1)^{\pi_S} = \boldsymbol{U}^2_1\ \underline{0}\ \boldsymbol{U}^3_1 = \underline{0}$

$(a_1\ \underline{2}\ \square\ a_2)^{\pi_S} = \boldsymbol{U}^3_1\ \underline{2}\ \square\ \boldsymbol{U}^2_1 = \underline{2}$

Church numerals at a are selected. Then, separation trees between Church numerals have the form $a' = 1 \cdot 1 \cdot \cdot \cdot 1$.

(6) $(\ )^{\pi_1} = (\ )\ f\ p$

$(\underline{0})^{\pi_1} = (\lambda fx.x)\ f\ p = p$

$(\underline{2})^{\pi_1} = (\lambda fx.f(fx))\ f\ p = f\ (f\ p)$

where every $\sim$ equivalence class consists of single λ-free term.

(7) Now we apply π_1 transformation to encode RHS. In (6) p and f should be related to the corresponding RHS. Given terms in the form of $\langle\langle\underline{0}\rangle, \phi(N)\rangle$ *and* $\langle\langle\underline{2}, \phi(M)\rangle,\phi(N)\rangle$, p is replaced by the function which selects the subterm at 2 and f by the function which selects subterms at $1 \cdot 2$ and 2 and constructs $\phi(S(A(M, N)))$. Then, the whole encoding π is a composition of $\pi_{a \cdot a'}$ and π_1; $\pi = \pi_1 \cdot \pi_{a \cdot a'}$.

$(\ )^{\pi 1} = (\ )\ [p := \boldsymbol{U}^2_2][f := \lambda b.(\phi(S)\ (\phi(A)\ \langle(($ $)\boldsymbol{U}^2_1\boldsymbol{U}^2_2), ((\ )\ \boldsymbol{U}^2_2)\rangle))]$, where ( ) means the input of a tuple of arguments.

$(\ )^{\pi} = (\ )\ z_1\ [z_1 := \boldsymbol{U}^2_1]\ z_2\ a_1\ a_2\ [z_2 := \boldsymbol{P}_2]\ [a_2 := \boldsymbol{U}^2_1]\ [a_1 := \boldsymbol{U}^3_1]\ f\ p\ [p := \boldsymbol{U}^2_2][f := \lambda b.(\phi(S)\ (\phi(A)\ \langle(($ $)\boldsymbol{U}^2_1\boldsymbol{U}^2_2), ((\ )\ \boldsymbol{U}^2_2)\rangle))]$

$\equiv (\ )\ \boldsymbol{U}^2_1\ \boldsymbol{P}_2\ \boldsymbol{U}^3_1\ \boldsymbol{U}^2_1\ (\lambda b.(\phi(S)\ (\phi(A)\ \langle(($ $)\boldsymbol{U}^2_1\boldsymbol{U}^2_2), ((\ )\ \boldsymbol{U}^2_2)\rangle)))\ ((\ )\boldsymbol{U}^2_2)$.

Then,

$\phi(A) = \lambda x.x\ \boldsymbol{U}^2_1\ \boldsymbol{P}_2\ \boldsymbol{U}^3_1\ \boldsymbol{U}^2_1\ (\lambda b.(\phi(S)\ (\phi(A)\ \langle(x\ \boldsymbol{U}^2_1\boldsymbol{U}^2_2), (x\ \boldsymbol{U}^2_2)\rangle)))\ (x\ \boldsymbol{U}^2_2)$

$= \boldsymbol{Y}\ (\lambda ax.x\ \boldsymbol{U}^2_1\ \boldsymbol{P}_2\ \boldsymbol{U}^3_1\ \boldsymbol{U}^2_1\ (\lambda b.\ \langle 2,\ (a\ <\ (x\ \boldsymbol{U}^2_1\boldsymbol{U}^2_2),\ (x\ \boldsymbol{U}^2_2)\rangle)\rangle)\ (x\ \boldsymbol{U}^2_2)$

Reductions $A(S(0), 0) \rightarrow S(A(0, 0)) \rightarrow S(0)$ are simulated as follows.

$\phi(A(S(0), 0)) = \phi(A)\ \phi_p(S(0), 0) = \phi(A)\ \langle\phi(S(0)), \phi(0)\rangle$

$= \phi(A)\ \langle\langle\underline{2}, \langle\underline{0}\rangle\rangle, \langle\underline{0}\rangle\rangle$

$\equiv \boldsymbol{Y}\ (\lambda ax.x\ \boldsymbol{U}^2_1\ \boldsymbol{P}_2\ \boldsymbol{U}^3_1\ \boldsymbol{U}^2_1\ (\lambda b.\langle 2,\ (a\ \langle(x\ \boldsymbol{U}^2_1\boldsymbol{U}^2_2),\ (x\ \boldsymbol{U}^2_2)\rangle)\rangle)\ (x\ \boldsymbol{U}^2_2)\ \langle\langle\underline{2}, \langle\underline{0}\rangle\rangle, \langle\underline{0}\rangle\rangle$

$\rightarrow^* \langle\underline{2}, (\phi(A)\ \langle\langle\underline{0}\rangle, \langle\underline{0}\rangle\rangle)\rangle$

$\rightarrow^* \langle\underline{2}, \langle\underline{0}\rangle\rangle\ (= \phi(S(0)))$

변 석 우
정보과학회논문지 : 시스템 및 이론
제 35 권 제 3 호 참조