Algebraic Semantics for Higher-order Functional-Logic Programming

Makoto Hamana
Doctral Program in Engineering, University of Tsukuba,
Tsukuba 305, Japan
E-mail: hamana@score.is.tsukuba.ac.jp

ABSTRACT

In this paper we give a semantics of higher-order functional-logic programming in the framework of typed universal algebra. The functional-logic language concerned here is an applicative term rewriting system in which there is no lambda abstraction mechanism. Therefore ordinary first-order narrowing can be used to solve higher-order query. The soundness and completeness of narrowing for both the initial and minimal applicative Herbrand model is shown. We discuss the requirements of "minimal" for applicative Herbrand model, which means that any elements in the domain must have denotations of some terms, is need to obtain the completeness of narrowing in the applicative case of functional-logic programming.

1. Introduction

Semantics of first-order functional-logic programming has been deeply investigated. Two approaches are known for giving semantics. One approach is based on term rewriting. In this approach, a program of functional-logic languages is regarded as a conditional term rewriting system [10, 1], a query as satisfiability of an equation with respect to the underlying equational theory. Since conditional narrowing is a sound and complete procedure for establishing convertibility [9], operational models of some existing functional-logic languages are based on (refined version of) conditional narrowing, e.g., ALF [8], Ev [6].

The other approach is to extend declarative semantics of logic programming languages. This approach was first described by Levi *et al.* [11] for their logic plus functional language K-LEAF. In this approach, a program is to define (partial) functions on the complete Herbrand universe that contains infinite data structures and solving a query is finding a substitution that is a solution for the query in the least complete Herbrand model. The language BABEL [14] also has the semantics along this approach. Sound and complete operational models for these languages were given.

Recently higher-order extensions of functional-logic languages are investigated. The language SFL [4][5], which is a higher-order extension of BABEL, has the form of applicative equations for a program. Two semantics for SFL were proposed in [4]. One is the "declarative semantics" of SFL which has essentially the semantics along the declarative approach described above, where a slight difference is that each element in the Herbrand universe has applicative form. The other is the "denotational

semantics" of SFL which is a also declarative approach essentially but the Herbrand universe is extended to the universe that contains function spaces because functions are used as data in higher-order programming.

The rewriting approach for semantics of higher-order functional-logic language is given in [15]. In this approach, a program is an applicative term rewriting system and the presented narrowing calculus for higher-order functional-logic programming is shown to be sound and complete for proving convertibility generated from applicative term rewriting, which is actually first-order.

The semantics presented in this paper is a further development of both SFL's "denotational" approach [4] and applicative term rewriting approach [15]. We treat a program as an applicative term rewriting system in operational semantics, i.e., first-order conditional narrowing is used to solve a higher-order query. For algebraic semantics, first we present the initial model in a class of all models of a program as the quotient of convertibility generated from the applicative term rewriting. Next we give another model, which is called "minimal applicative Herbrand model". The carrier of this model contains function spaces, which is similar to the "denotational semantics" of SFL. The important difference compared with it is that our semantics is sound and complete for operational model of narrowing unlike SFL's "denotational semantics". This desirable result comes from the minimality of applicative Herbrand model in our semantics defined in Section 5.

The paper is organized as follows. In Section 2, we define the syntax of a functional-logic language Ev. In Section 3, we construct the initial model, called quotient applicative term model, of Ev by a quotient of congruence generated from term rewriting. In Section 4, we review first-order conditional narrowing and apply its soundness and completeness results to the quotient applicative term model. In Section 5, we construct the minimal applicative Herbrand model and discuss why minimality is required in order to obtain the completeness of solving higher-order query under an applicative term rewriting system. In Section 6, we establish soundness and completeness of operational model of narrowing for the minimal applicative Herbrand model. In Section 7, we conclude the paper.

2. Syntax of a Higher-order Functional-Logic Language Ev

In this section we define syntax of a higher-order functional-logic language Ev. The language of Ev is generated from many-sorted signature. We use the notation and terminology of higher type universal algebra [12].

Definition 2.1 [**Many-sorted Algebra**]
A set S of *sorts* and B of *type basis* are any nonempty set. A set S of sort is a *type structure* S over a type basis B if $S = B \cup \{\sigma \to \tau \mid \sigma, \tau \in S\}$. Each element $\sigma \in S$ is termed a *type*; each element $\sigma \in B$ and each element of the form $\sigma \to \tau \in S$ are termed a *base type* and *function type* respectively. For simplicity, we assume a type

basis B is always a singleton set $\{o\}$ where the type o is the only base type. The theory developing in this paper will be straightforwardly extended to the case that B is any nonempty set. An S-sorted signature $\mathcal{F}$ is an $S^* \times S$ indexed collection of disjoint sets $\mathcal{F} = \{\mathcal{F}^{w,\tau} \mid w \in S^*, \tau \in S\}$ where S^* denotes the set of all sequences generated from S.

An *S-sorted $\mathcal{F}$-algebra* is a pair $\langle A, \mathcal{F}_\mathcal{A}\rangle$, consisting of an S-indexed collection $A = \{A^\tau | \tau \in S\}$ of non-empty carrier sets A^s and an indexed collection $\mathcal{F}_\mathcal{A} = \{\mathcal{F}_\mathcal{A}^{w,s} \mid w \in S^*, \tau \in S\}$ of sets of operations. For each sequence of sorts $w = \sigma_1 \ldots \sigma_n \in S^*$ and each sort $s \in S$, $\mathcal{F}_\mathcal{A}^{w,\tau} = \{f_\mathcal{A} \mid f \in \mathcal{F}^w, \tau \in S\}$, where $f_\mathcal{A} : A^{\sigma_1} \times \ldots \times A^{\sigma_n} \to A^\tau$. An *$S$-sorted homomorphism* from an S-sorted algebra $\mathcal{A}$ to $\mathcal{B}$ is an S-indexed family of mapping $\phi^\sigma : \mathcal{A} \to \mathcal{B}$ such that for each $\sigma \in S$, each $w = \sigma_1, \ldots, \sigma_n \in S^*$, any function symbol $f \in \mathcal{F}^{w,\sigma}$ and any $a_1, \ldots, a_n \in \mathcal{A}^w$, $\phi^\sigma(f_\mathcal{A}(a_1, \ldots, a_n)) = f_\mathcal{B}(\phi^{\sigma_1}, \ldots, \phi^{\sigma_n})$. An *$S$-typed signature* $\mathcal{F}$ is an S-sorted signature $\mathcal{F}$ such that for each function type $(\sigma \to \tau) \in S$ we have an application symbol $ap^{\sigma\to\tau} \in \mathcal{F}^{\sigma\to\tau\sigma,\tau}$. Let $\mathcal{F}$ be an S-typed signature. An $\mathcal{F}$-algebra $\mathcal{A}$ is an *S-typed algebra* if for each function type $(\sigma \to \tau) \in S$, the following conditions are satisfied:

- $\mathcal{A}^{\sigma\to\tau} \subseteq (\mathcal{A}^\sigma \to \mathcal{A}^\tau)$,
- $\mathrm{ap}_\mathcal{A}^{\sigma\to\tau} : \mathcal{A}^{\sigma\to\tau} \times \mathcal{A}^\sigma \to \mathcal{A}^\tau$, $\mathrm{ap}_\mathcal{A}^{\sigma\to\tau}(f, t) = f(t)$.

Definition 2.2 [**Applicative Terms**]
Let S be a type structure. We will use a notation o^n which denotes n-curried function type $o \to \ldots \to o$ where $\to$ is right associative. We assume that S-typed signature $\mathcal{F}$ is a disjoint union of CON, FUN and $\{\{\mathrm{ap}^{\sigma\to\tau}\} \mid \sigma, \tau \in S\}$ where $\mathrm{FUN} = \{\mathrm{FUN}^\sigma \mid \sigma \in S\}$ and $\mathrm{CON} = \{\mathrm{CON}^\sigma \mid n \in \mathbb{N}, \sigma = o^n \to o\}$; FUN^σ is a set of defined function symbols with type σ and CON^σ is a set of *constructor symbols* with type σ. A defined function symbol is called *function symbol* for short, hereafter.

A set VAR of variables is an S-indexed collection of disjoint sets $\mathrm{VAR} = \{\mathrm{VAR}^s \mid s \in S\}$. An *applicative term* t of *type* σ, denoted by $t : \sigma$, is obtained by applying the following rules finitely many times:

$$\frac{}{c : \sigma} \quad \text{for each } c \in \mathrm{CON}^\sigma \qquad \frac{}{f : \sigma} \quad \text{for each } f \in \mathrm{FUN}^\sigma$$

$$\frac{}{x : \sigma} \quad \text{for each } x \in \mathrm{VAR}^\sigma \qquad \frac{s : \sigma \to \tau, \quad t : \sigma}{\mathrm{ap}^{\sigma\to\tau}(s, t) : \tau}$$

The set $\mathcal{AT}(\mathrm{VAR})$ of all applicative terms is an S-indexed collection $\mathcal{AT}(\mathrm{VAR}) = \{\mathcal{AT}(\mathrm{VAR})^\sigma \mid \sigma \in S\}$ of disjoint sets $\mathcal{AT}(\mathrm{VAR})^\sigma$ of all applicative terms of a type σ. We use curried notation for an applicative terms, for example, we write $f\ 0\ 1$ as $\mathrm{ap}(\mathrm{ap}(f, 0), 1)$. A term that does not contain any occurrence of a defined function symbol is called *data term*.

An *assignment* ϕ is a mapping from a set of variables X to an S-sorted $\mathcal{F}$-algebra $\mathcal{A}$ and often presented as $\{x_1 \mapsto t_1, \cdots, x_n \mapsto t_n\}$, where $[\sigma](x_i) = t_i$ for each $x_i \in$

$X, i = 1, \cdots, n$. Especially, an assignment from X to a subset $\mathcal{AT}(Y)$ of terms is called *substitution*. The assignment ϕ is uniquely extended to a homomorphism $\phi^\#$ from $\mathcal{AT}(X)$ to $\mathcal{A}$ as follows:

$$\begin{aligned}
\phi^\#(f) &= f_\mathcal{A}, & f \in \textsc{Fun},\\
\phi^\#(c) &= c_\mathcal{A}, & c \in \textsc{Con},\\
\phi^\#(x) &= \phi(x), & x \in \textsc{Var},\\
\phi^\#(\mathrm{ap}^{\sigma\to\tau}(t_1, t_n)) &= \mathrm{ap}^{\sigma\to\tau}_\mathcal{A}(\phi^\#(t_1), \phi^\#(t_n)).
\end{aligned}$$

Definition 2.3 [**Applicative Equations**]
Let $s, t \in \mathcal{AT}(X)^\sigma$ and $s_1, \ldots, s_n \in \mathcal{AT}(X)^o$ where $\sigma \in S$ and $X \subseteq \textsc{Var}$. A *conditional equation* is a formula of the form

$$\forall X(s = t \Leftarrow s_1 = \mathrm{true}, \cdots, s_n = \mathrm{true}).$$

We always assume the set X of quantified variables is exactly the set of variables occurring in the body of formula. We often write the quantified variables X as the sequence $x_1 : \sigma_1, \ldots, x_n : \sigma_n$ which denotes $x_1 \in \textsc{Var}^{\sigma_1}, \ldots, x_n \in \textsc{Var}^{\sigma_n}$. An *existentially quantified equation* is a formula of the form

$$\exists X(s = t).$$

A conditional equation $\forall X(s = t \Leftarrow s_1 = t_1, \cdots, s_n = t_n)$ is *valid* in an S-sorted $\mathcal{F}$-algebra $\mathcal{A}$, or $\mathcal{A}$ is a model of the conditional equation, denoted by $\mathcal{A} \models \forall X(s = t \Leftarrow s_1 = t_1, \cdots, s_n = t_n)$, if for all assignments $\alpha : X \to \mathcal{A}$, $\alpha^\#(s) = \alpha^\#(t)$ whenever $\alpha^\#(s_i) = \alpha^\#(t_i)$ for all $i = 1, \cdots, n$. An existentially quantified equation $\exists X(s = t)$ is valid in $\mathcal{A}$, denoted by $\mathcal{A} \models \exists X(s = t)$, if there exists an assignment $\alpha : X \to \mathcal{A}$, called a *witness*, such that $\alpha^\#(s) = \alpha^\#(t)$. An $\mathcal{F}$-algebra $\mathcal{A}$ is a model of a set of conditional equations E if for all conditional equations $\forall X(s = t \Leftarrow s_1 = t_1, \cdots, s_n = t_n)$ in E, $\mathcal{A} \models \forall X(s = t \Leftarrow s_1 = t_1, \cdots, s_n = t_n)$. The notation $E \models \forall X(s = t)$ means that for any model $\mathcal{A}$ of E, $\mathcal{A} \models \forall X(s = t)$. Likewise the notation $E \models \exists X(s = t)$ means that for any model $\mathcal{A}$ of E, $\mathcal{A} \models \exists X(s = t)$.

Next we define *strict equality*. We intend that "data" in functional-logic programming are data terms. Therefore we need a computable equality on data terms. We define "built-in" axioms for any programs to express such an equality in the following.

Hereafter we assume an S-sorted signature for functional-logic programs is equipped with symbols $\mathrm{true} \in \textsc{Con}^o$ and $\mathrm{steq}, \& \in \textsc{Fun}^{o\to o\to o}$.

Definition 2.4 [**Strict Equality**]
We write "&" as an infix operator that is right associative. We define a set of equations

$\mathcal{R}_{\text{CON}}$ as follows:

$$\begin{array}{lll}
\mathcal{R}_{\text{CON}} & \stackrel{def}{=} & \{\forall\varnothing(\text{steq } c\ c = \text{true}) \mid c \in \text{CON}^o\} \\
& \cup & \{\forall x_1, \cdots, x_n, y_1, \cdots, y_n : o \\
& & \quad (\text{steq}(d\ x_1 \cdots x_n)(d\ y_1 \cdots y_n)) \\
& & \quad\quad = (\text{steq } x_1\ y_1)\ \&\ \cdots\ \&\ (\text{steq } x_n\ y_n)) \mid d \in \text{CON}^{o^n \to o}\} \\
& \cup & \{\forall x : o(\text{true}\ \&\ x = x)\},
\end{array}$$

We call an equation of the form $(\text{steq } s\ t) = \text{true}$, where the terms s and t do not contain any occurrence of steq, *strict equation* and the induced equality from $\mathcal{R}_{\text{CON}}$ *strict equality.*

Definition 2.5 [**Functional-Logic Program**]
A set $\mathcal{R}$ of applicative conditional equations is a *program* if for each equation $\forall X(l = r \Leftarrow c)$ in $\mathcal{R}$ the following conditions are satisfied:

1. $\mathcal{R}$ contains the axioms for strict equality $\mathcal{R}_{\text{CON}}$.
2. A variable occurring in r always occurs in l.
3. l is linear, i.e., it does not contain multiple occurrences of the same variables.
4. l is the form $f\ t_1 \cdots t_n$ where $f \in \text{FUN}^{\sigma_1 \to \cdots \to \sigma_n \to \sigma}$ and the terms $t_1 : \sigma_1, \cdots, t_n : \sigma_n$ do not contain function symbols, i.e., each term is built from only variables and constructors.
5. $\mathcal{R}$ is non-ambiguous. That is, in this case, for any two different equations $\forall X_1(l_1 = r_1 \Leftarrow c_1)$ and $\forall X_2(l_2 = r_2 \Leftarrow c_2)$ in $\mathcal{R}$, l_1 and l_2 are not unifiable.
6. All equations in the conditional part c are strict equations.

Note that any functional-logic program as defined above is confluent because the conditional term rewriting system obtained from a functional-logic program by interpreting the conditional equations as conditional rewrite rules is an orthogonal III_n[a] type-2[b] conditional term rewriting system [1].

Definition 2.6 [**Query and Execution of Functional-Logic Program**]
Let $\mathcal{R}$ be a program and $s, t \in \mathcal{AT}(X)$ where X is a finite subset of VAR. A *query* is an existentially quantified strict equation of the form

$$\exists X(\text{steq } s\ t = \text{true}).$$

[a] The terminology "III_n" means that each equations in the conditions is the form $t = d$ where d is a closed normal form with respect to the system obtained by removing the conditions [1].

[b] The terminology "type-2" means that a conditional term rewriting system $\mathcal{R}$ may only contain extra variables in the conditions [13]

Execution of a functional-logic program $\mathcal{R}$ is to prove that the validity of the query

$$\mathcal{R} \models \exists X(\text{steq } s\ t = \text{true})$$

is valid by obtaining a witness of the existential equation.

Example 2.7
Let the symbols plus $\in \text{FUN}^{o\to o\to o}$, map $\in \text{FUN}^{(o\to o)\to o\to o}$ and $\text{Nil}, 0 \in \text{CON}^{o}$, S $\in \text{CON}^{o\to o}$, Cons $\in \text{CON}^{o\to o\to o}$. Following is an example of program consisting of an addition function on encoded natural numbers and "map" function over list.

$$\mathcal{R} = \left\{ \begin{array}{rl} \forall y : o & (\text{plus } 0\ y = y) \\ \forall x, y : o & (\text{plus } (\text{S } x)\ y = \text{S } (\text{plus } x\ y)) \\ \forall f : o \to o & (\text{map } f\ \text{Nil} = \text{Nil}) \\ \forall f : o \to o, x, xs : o & (\text{map } f\ (\text{Cons } x\ xs) \\ & \quad = \text{Cons } (f\ x)\ (\text{map } f\ xs)) \end{array} \right\}.$$

Under the program $\mathcal{R}$, examples of a query are:

$$\mathcal{R} \models \exists z : o(\text{steq } (\text{plus } (\text{S } 0)\ z)\ (\text{S } z)) = \text{true},$$

or

$$\mathcal{R} \models \exists f : o \to o\ (\text{steq } (\text{map } f\ [0, (\text{S } 0)])\ [\text{S } 0, \text{S } (\text{S } 0)] = \text{true}).$$

Operational and algebraic semantics presented in this paper will be concerned with this kind of query where solutions are not only of base types but also of function types.

3. Quotient Applicative Term Model

In this section we construct the initial model of Ev by a quotient of congruence generated from term rewriting. For an program $\mathcal{R}$ conditional term rewriting [1] is defined.

Definition 3.1 [Conditional Term Rewriting]
A rewrite relation $\to_{\mathcal{R}}$ on applicative terms $\mathcal{AT}(\text{VAR})$ for a program $\mathcal{R}$ between applicative terms are inductively defined as follows:

$$\begin{array}{rcl} \to_{\mathcal{R}_0} & \stackrel{def}{=} & \varnothing, \\ \to_{\mathcal{R}_{n+1}} & \stackrel{def}{=} & \{\langle C[l\theta], C[r\theta]\rangle \mid \forall X(l = r \Leftarrow s_1 = \text{true}, \ldots, s_k = \text{true}) \in \mathcal{R}, \\ & & \qquad \theta : X \to \mathcal{AT}(\text{VAR}),\ C \text{ is a context}, \\ & & \qquad s_i\theta \to^*_{\mathcal{R}_n} \text{true for every } i = 1, \ldots, k\}, \\ \to_{\mathcal{R}} & \stackrel{def}{=} & \bigcup_{n\in\mathbb{N}} \to_{\mathcal{R}_{n+1}}. \end{array}$$

Here $\rightarrow^*$ denotes the reflexive and transitive closure of $\rightarrow$. We can easily check rewriting by $\rightarrow_{\mathcal{R}}$ preserves sort, i.e., if $s \in \mathcal{AT}(\text{VAR})^\sigma$ and $s \rightarrow^*_{\mathcal{R}} t$ then $t \in \mathcal{AT}(\text{VAR})^\sigma$.

Definition 3.2 [**Quotient Applicative Term Algebra**]
Let $\mathcal{R}$ be a program. Define the equivalence relation $\leftrightarrow^*_{G(\mathcal{R})} \stackrel{def}{=} \leftrightarrow^*_{\mathcal{R}} \cap \mathcal{AT}(\varnothing) \times \mathcal{AT}(\varnothing)$ where $\leftrightarrow^*_{\mathcal{R}}$ is the reflexive, transitive and symmetric closure of $\rightarrow^*_{\mathcal{R}}$. The $\leftrightarrow^*_{\mathcal{R}}$-equivalence class of a term $t \in \mathcal{AT}(\varnothing)$ is denoted by $[t]$, namely $[t] \stackrel{def}{=} \{s \in \mathcal{AT}(\varnothing) \mid s \leftrightarrow^*_{G(\mathcal{R})} t\}$. The *quotient applicative term algebra* $\mathcal{AQ}_{\mathcal{R}}$ is a pair $\langle \mathcal{AT}(\varnothing)/\!\leftrightarrow^*_{G(\mathcal{R})}, \mathcal{F}_{\mathcal{AQ}_{\mathcal{R}}} \rangle$, consisting of an S-indexed collection $\mathcal{AT}(\varnothing)/\!\leftrightarrow^*_{G(\mathcal{R})} = \{\mathcal{AT}(\varnothing)^\sigma/\!\leftrightarrow^*_{G(\mathcal{R})} \mid \sigma \in S\}$ of carrier sets and an indexed collection $\mathcal{F}_{\mathcal{AQ}_{\mathcal{R}}} = \{\mathcal{F}^{w,\sigma}_{\mathcal{AQ}_{\mathcal{R}}} \mid w \in S^*, \sigma \in S\}$ of operations. For each sequence of sorts $w = \sigma_1 \ldots \sigma_n \in S^*$ and each sort $\sigma \in S$,

$$\begin{aligned}
\mathcal{F}^{w,\sigma}_{\mathcal{AQ}_{\mathcal{R}}} \stackrel{def}{=} \{ f_{\mathcal{AQ}_{\mathcal{R}}} \mid\ & f \in \mathcal{F}^{w,\sigma}, \\
& f_{\mathcal{AQ}_{\mathcal{R}}} : (\mathcal{AT}(\varnothing)^{\sigma_1}/\!\leftrightarrow^*_{G(\mathcal{R})}) \times \ldots \times (\mathcal{AT}(\varnothing)^{\sigma_n}/\!\leftrightarrow^*_{G(\mathcal{R})}) \\
& \qquad \rightarrow (\mathcal{AT}(\varnothing)^{\sigma}/\!\leftrightarrow^*_{G(\mathcal{R})}) \\
& f_{\mathcal{AQ}_{\mathcal{R}}}([t_1], \cdots, [t_n]) = [f(t_1, \cdots, t_n)] \}.
\end{aligned}$$

Note that the applicative term algebra $\mathcal{AT}(\varnothing)$ a many-sorted term algebra. So from the result of many-sorted algebra [3], the quotient $\mathcal{AQ}_{\mathcal{R}}$ is the initial model of the class of all models of $\mathcal{R}$. Moreover, a version of Herbrand theorem [3] also holds. We state an applicative version of their result in the following.

Theorem 3.3 [**Herbrand's Theorem for Applicative Equations**]
The quotient applicative term algebra $\mathcal{AQ}_{\mathcal{R}}$ is a model of a program $\mathcal{R}$. Moreover the following propositions hold:

$$\begin{aligned}
& \mathcal{R} \models \exists X(\text{steq } s\ t = \text{true}) \\
\Leftrightarrow\ & \mathcal{AQ}_{\mathcal{R}} \models \exists X(\text{steq } s\ t = \text{true}) \\
\Leftrightarrow\ & \mathcal{AQ}_{\mathcal{R}} \models \forall \varnothing((\text{steq } s\ t)\theta = \text{true}) \quad \text{for some } \theta : X \rightarrow \mathcal{AT}(\varnothing) \\
\Leftrightarrow\ & (\text{steq } s\ t)\theta \rightarrow^*_{\mathcal{R}} \text{true}.
\end{aligned}$$

4. Operational Model of Narrowing

We use first-order narrowing as an operational model of Ev. First-order narrowing can also work in higher-order functional-logic language Ev. Because, as noted before, applicative terms are syntactically first-order terms.

Definition 4.1 [**Conditional Narrowing**]
Let $\mathcal{R}$ be a program and S, T be terms that are conjunction (by the function symbol &) of strict equations or true's. Narrowing relation $\rightsquigarrow$ on $\mathcal{AT}(\text{VAR})$ is defined as follows: $S \rightsquigarrow_\theta T$ if there exists a non-variable position $u \in \mathcal{P}os(S)$, a new variant (with

respect to variable-renaming) $\forall X(l = r \Leftarrow \text{steq } s_1\ t_1 = \text{true}, \cdots, \text{steq } s_n\ t_n = \text{true})$ of a conditional equation in $\mathcal{R}$, and a substitution θ such that

- θ is a most general unifier of $S_{|u}$ and l,
- $T = (S[r]_u \ \& \ (\text{steq } s_1\ t_1) \ \& \ \cdots \ \& \ (\text{steq } s_n\ t_n))\theta$.

We write $S \rightsquigarrow^*_\theta T$ if there exists a narrowing derivation $(S \equiv)S_1 \rightsquigarrow_{\theta_1} S_2 \rightsquigarrow_{\theta_2} \cdots \rightsquigarrow_{\theta_{n-1}} S_n(\equiv T)$ such that $\theta = \theta_{n-1} \circ \cdots \circ \theta_2 \circ \theta_1$.

Conditional narrowing defined above is sound and complete for rewriting presented below. We use the completeness result of conditional narrowing with respect to sufficiently normalizable solution for orthogonal type-2 conditional term rewriting systems with strict equality obtained by Ida and Okui [9]. We adapt their result to our setting.

Theorem 4.2 [Soundness and Completeness of Conditional Narrowing for Rewriting]
Let $\mathcal{R}$ be a program and $\exists X(\text{steq } s\ t = \text{true})$ a query. Conditional narrowing is sound for rewriting, i.e., if $(\text{steq } s\ t) \rightsquigarrow^*_\theta \text{true}$ where $\theta : X \to \mathcal{AT}(Z)$ then $(\text{steq } s\ t)\theta \to^*_{\mathcal{R}} \text{true}$. Moreover conditional narrowing is complete for rewriting, i.e., for arbitrary normalized substitution[c] $\theta : X \to \mathcal{AT}(Z)$, if $(\text{steq } s\ t)\theta \to^*_{\mathcal{R}} \text{true}$ then there exists a narrowing derivation $(\text{steq } s\ t) \rightsquigarrow^*_{\theta'} \text{true}$ where $\theta' : X \to \mathcal{AT}(Y)$ such that $\rho \circ \theta' = \theta$ for any $\rho : Y \to \mathcal{AT}(Z)$.

Combining Theorems 3.3 and 4.2, we get the soundness and completeness of narrowing with respect to the quotient applicative term model.

Theorem 4.3
Let $\mathcal{R}$ be a program and $\exists X(\text{steq } s\ t = \text{true})$ a query. The query $\mathcal{AQ}_\mathcal{R} \models \exists X(\text{steq } s\ t = \text{true})$ is valid with a witness $\gamma : X \to \mathcal{AQ}_\mathcal{R}$ such that for each $x \in X$, an equivalence class $\gamma(x)$ contains a normal form if and only if there exits a narrowing derivation $(\text{steq } s\ t) \rightsquigarrow^*_\theta \text{true}$ with a normalized substitution $\theta : X \to \mathcal{AT}(Y)$. In the both direction of the implication, $\mathcal{AQ}_\mathcal{R}[\![\_]\!] \circ \rho \circ \theta = \gamma$ holds for any substitution $\rho : Y \to \mathcal{AT}(\varnothing)$ where the mapping $\mathcal{AQ}_\mathcal{R}[\![\_]\!]$ is the unique homomorphism from $\mathcal{AT}(\varnothing)$ to $\mathcal{AQ}_\mathcal{R}$.

5. Minimal Applicative Herbrand Model

We construct another model, called *minimal applicative Herbrand model*. This model provides the following natural meaning of a higher-order functional-logic program:

[c] A substitution θ is called normalized if for all $x \in X$, $x\theta$ is a normal form.

- *Data* in higher-order functional-logic programming are (infinite) data terms and *functions* (not function symbols) defined by a program of an applicative term rewriting system.
- *Functions* in a higher-order functional-logic programming are partial functions on the domain of *data*.
- A *query* of a higher-order functional-logic programming is an existentially quantified strict equation which can contain function type variable as quantified variables.

Soundness and completeness of narrowing with respect to the minimal applicative Herbrand model shows that such intuitive explanations are completely correct.

Definition 5.1 [**Applicative Herbrand Universe**]
An applicative Herbrand universe H^o is a set of finite and infinite binary trees that is deduced from the following deduction rules.

$$\frac{}{c : \sigma} \quad \text{for each } c \in \mathrm{CON}^\sigma \qquad \frac{}{\bot : o} \qquad \frac{s : \sigma \to \tau, \quad t : \sigma}{\mathrm{ap}^{\sigma\to\tau}(s,t) : \tau}$$

$$\mathrm{H}^o \stackrel{def}{=} \{t \mid t : o \text{ is derived by the above rules}\}.$$

The set of finite trees in H^o exactly corresponds to the set of all ground data terms with the type o by identifying a tree as a term. Applying the above deduction rules infinitely many times, infinite trees are deduced.

Definition 5.2 [**Applicative Herbrand Algebra**]
Let $\mathcal{F} = \mathrm{CON} \cup \mathrm{FUN} \cup \{\{\mathrm{ap}^{\sigma\to\tau}\} \mid \sigma, \tau \in S\}$ be an S-typed signature. An S-typed $\mathcal{F}$-algebra $\mathcal{A} = \langle H, \mathcal{F}_\mathcal{A}\rangle$ is an *applicative Herbrand algebra* if its carrier is an S-indexed collection $\mathrm{H} = \{\mathrm{H}^\sigma \mid \sigma \in S\}$ where the base type carrier is H^o and the function type carrier $\mathrm{H}^{\sigma\to\tau}$ is the set $(\mathrm{H}^\sigma \to \mathrm{H}^\tau)$ of all continuous functions for each function type $\sigma \to \tau$. The operations $\mathcal{F}_\mathcal{A}$ satisfy the following:

$$\mathrm{CON}_\mathcal{A}^{o^n\to o} = \{c_\mathcal{A} \mid c \in \mathrm{CON}^{o^n\to o}, c_\mathcal{A} \in \mathrm{H}^{o^n\to o},\ c_\mathcal{A}\, t_1 \cdots t_n = c\, t_1 \cdots t_n\},$$
$$\mathrm{FUN}_\mathcal{A}^{\sigma\to\tau} = \{f_\mathcal{A} \mid f \in \mathrm{FUN}^{\sigma\to\tau},\ f_\mathcal{A} \in \mathrm{H}^{\sigma\to\tau},\ f_\mathcal{A} \text{ is a continuous function }\}$$

for each $\sigma, \tau \in S$.

In the definition of applicative Herbrand algebra, the operations for function symbols are not specified, only defined as some continuous functions. So an S-typed algebra $\mathcal{A}$ that can be a model of a program is an algebra that has suitable set of functions for $\mathrm{FUN}_\mathcal{A}$ corresponding to the program.

Definition 5.3
Let $\mathcal{A}$ be an applicative Herbrand algebra. For each $\sigma \in S$ the order $\sqsubseteq_{\mathrm{H}^\sigma}$ on H^σ is defined as:

- For the base type o, $\sqsubseteq_{\mathrm{H}^o}$ is a usual approximation ordering on trees,
- For each function type $(\sigma \to \tau) \in S$, $\sqsubseteq_{\mathrm{H}^{\sigma\to\tau}}$ is a usual pointwise ordering on functions.

Define HA as the class of all applicative Herbrand algebra and an order $\sqsubseteq_{\mathrm{HA}}$ on HA as:

$$\mathcal{A} \sqsubseteq_{\mathrm{HA}} \mathcal{B} \stackrel{def}{\Longleftrightarrow} f_{\mathcal{A}} \sqsubseteq_{\mathrm{H}^\sigma} f_{\mathcal{B}} \text{ for each } f \in \mathcal{F}^\sigma.$$

Proposition 5.4
For each $\sigma \in S$, $\langle \mathrm{H}^\sigma, \sqsubseteq_{\mathrm{H}^\sigma} \rangle$ and $\langle \mathrm{HA}, \sqsubseteq_{\mathrm{HA}} \rangle$ are algebraic[d] cpos.

We denote the least elements of the cpos H and HA by $\bot_{\mathrm{H}}$ and $\bot_{\mathrm{HA}}$.

Definition 5.5 [Construction of the Least Applicative Herbrand Model]
Let $\mathcal{F}$ be an S-typed signature and $\mathcal{R}$ be a program. Define an operator $\mathrm{T}_{\mathcal{R}} : \mathrm{HA} \to \mathrm{HA}$ as:

$$\begin{aligned}\mathrm{T}_{\mathcal{R}}(\mathcal{A}) \stackrel{def}{=} \langle \mathrm{H}, \{\{\mathrm{ap}_{\mathcal{A}}^{\sigma\to\tau}\} \mid \sigma,\tau \in S\} \cup \mathrm{CON}_{\mathcal{A}} \\ \cup \{\{\Phi_{\mathcal{A}}(f) \in \mathrm{H}^\sigma \mid f \in \mathrm{FUN}^\sigma\} \mid \sigma \in S\}\rangle\end{aligned}$$

where for each $f \in \mathrm{FUN}^{\sigma_1\to\ldots\to\sigma_n\to\tau}$, a mapping $\Phi_{\mathcal{A}}(f) : \mathrm{H}^{\sigma_1}\to\ldots\to\mathrm{H}^{\sigma_n} \to \mathrm{H}^\tau$ is defined as:

$$\Phi_{\mathcal{A}}(f)(h_1)\cdots(h_n) \stackrel{def}{=} \begin{cases} \delta^\#(r) & \text{if } \exists(\forall X(f\ l_1\ \cdots\ l_n = r \\ & \quad \Leftarrow s_1 = \mathrm{true}, \cdots, s_n = \mathrm{true}) \in \mathcal{R}, \\ & \exists \delta : X \to \mathcal{A}, \\ & \forall i \in \{1,\cdots,n\}; h_i = \delta^\#(l_i), \\ & \forall i \in \{1,\cdots,m\}; \delta^\#(s_i) = \mathrm{true}, \\ \bot_{\mathrm{H}} & \text{otherwise.} \end{cases}$$

The operator $\mathrm{T}_{\mathcal{R}}$ is a continuous function on HA. This can be proved by showing that each construct of $\mathrm{T}_{\mathcal{R}}$ are continuous. By the fixpoint theorem on cpos, $\mathrm{T}_{\mathcal{R}}$ has the least fixpoint expressed as follows:

$$\mathcal{AH}_{\mathcal{R}}' \stackrel{def}{=} \bigsqcup_{i\in N} \mathrm{T}_{\mathcal{R}}{}^i(\bot_{\mathrm{HA}}).$$

[d] A domain D is *algebraic* if for all $x \in D$, the set $S_x = \{z \mid z \sqsubseteq x \text{ and } z \text{ is compact}\}$ is directed and $x = \bigsqcup S_x$.

Theorem 5.6
$\mathcal{AH}_{\mathcal{R}}{}'$ is the least (with respect to $\sqsubseteq_{\mathrm{HA}}$) applicative Herbrand model of a program $\mathcal{R}$.
Proof It is proven in the same way as the first-order case [2],[7].

The carrier of $\mathcal{AH}_{\mathcal{R}}{}'$ contains many elements that cannot be denoted by applicative terms. For example, consider a program $\mathcal{R} = \{\forall x : o(\mathrm{id}\ x = x)\}$ and a query

$$\mathcal{R} \models \exists f : o \to o(\mathrm{steq}\ (f\ 1)\ 1 = \mathrm{true})$$

where the signature $\mathcal{F} = \textsc{Con} \cup \textsc{Fun}$ and $\textsc{Con}^o = \{1\}, \textsc{Fun}^{o\to o} = \{\mathrm{id}\}$. A witness of the above query in $\mathcal{AH}_{\mathcal{R}}{}'$ is $\xi = \{f \mapsto \mathrm{id}_{\mathcal{AH}_{\mathcal{R}}{}'}\} : \{f\} \to \mathcal{AH}_{\mathcal{R}}{}'$ where $\mathrm{id}_{\mathcal{AH}_{\mathcal{R}}{}'} \in \mathrm{H}^{o\to o}$ is an identity function on H^o. The witness ξ can be "represented"as a substitution $\theta = \{f \mapsto \mathrm{id}\}$. This means that the system of the functional-logic language Ev can return an answer for the query as the form of the applicative term.

On the other hand, there exits another witness for the above query, which is $\zeta = \{f \mapsto \alpha\}$ where a function $\alpha \in \mathrm{H}^{o\to o}$ defined as $\alpha(x) = 1 \in \mathrm{H}^o$. But the witness ζ can not be represented by a substitution because the function $\alpha \in \mathrm{H}^{o\to o}$ is not a denotation of any applicative term constructed from the signature $\mathcal{F}$.

This unsatisfactory result is caused by the definition of the carrier of function types. In this example, since the carrier $\mathrm{H}^{o\to o}$ is defined as the set of *all* continuous functions on $(\mathrm{H}^o \to \mathrm{H}^o)$, α is an element of $\mathrm{H}^{o\to o}$, so α is a witness of that query. However, in order to obtain the answer substition corresponding the witness of queries, we will exclude such α from the admissible witnesses that cannot be represented by substitutions to applicative terms. By minimizing the carriers of $\mathcal{AH}_{\mathcal{R}}{}'$ that only contain elements that can be denoted by applicative terms, this suitable model is obtained.

Definition 5.7
Let $\mathcal{R}$ be a program. The S-typed $\mathcal{F}$-algebra $\mathcal{AH}_{\mathcal{R}}$ is the homomorphic image of the unique homomorphism from $\mathcal{AT}(\varnothing)$ to $\mathcal{AH}_{\mathcal{R}}{}'$.

As the consequence of Theorem 5.6, we obtain the following.

Corollary 5.8
$\mathcal{AH}_{\mathcal{R}}$ is a model of a program $\mathcal{R}$.

We call the $\mathcal{F}$-algebra $\mathcal{AH}_{\mathcal{R}}$ *minimal applicative Herbrand model.* The terminology "minimal" comes from the notion of minimal algebra [12] which is also called term generated in the literature.

6. Soundness and Completeness

In this section we show that the minimal applicative Herbrand model is sound and complete for the operational model of narrowing. The soundness and completeness results express the correspondence between a witness in minimal applicative Herbrand model and a computed substitution by narrowing.

Theorem 6.1 [**Soundness of Narrowing for $\mathcal{AH}_\mathcal{R}$**]
Let $\mathcal{R}$ be a program and $\exists X(\text{steq } s\ t = \text{true})$ a query.

$$\begin{array}{rl} & (\text{steq } s\ t) \rightsquigarrow_\theta^* \text{true} \\ \Rightarrow & \mathcal{AH}_\mathcal{R} \models \exists X(\text{steq } s\ t = \text{true}) \\ & \quad \text{with a witness } \delta : X \to \mathcal{AH}_\mathcal{R} \end{array}$$

where $\theta : X \to \mathcal{AT}(Y)$ and $\delta = \mathcal{AH}_\mathcal{R}[\![\_]\!] \circ \rho^\# \circ \theta$ for arbitrary substitution $\rho : Y \to \mathcal{AT}(\varnothing)$. Here $\mathcal{AH}_\mathcal{R}[\![\_]\!]$ denotes the unique homomorphism form $\mathcal{AT}(\varnothing)$ to $\mathcal{AH}_\mathcal{R}$.
Proof Suppose $(\text{steq } s\ t) \rightsquigarrow_\theta^* \text{true}$. Then by the soundness of narrowing for rewriting, for any substitution $\rho : Y \to \mathcal{AT}(\varnothing)$, $(\text{steq } s\ t)\theta\rho \to_\mathcal{R}^* \text{true}$. So $\mathcal{AQ}_\mathcal{R}[\![(\text{steq } s\ t)\theta\rho]\!] = \mathcal{AQ}_\mathcal{R}[\![\text{true}]\!]$. Let $\phi : \mathcal{AQ}_\mathcal{R} \to \mathcal{AH}_\mathcal{R}$ be the unique homomorphism. Since $\mathcal{AH}_\mathcal{R}[\![\_]\!] = \phi \circ \mathcal{AQ}_\mathcal{R}[\![\_]\!]$,

$$\mathcal{AH}_\mathcal{R}[\![\_]\!] \circ \rho^\# \circ \theta\,(\text{steq } s\ t) = \phi(\mathcal{AQ}_\mathcal{R}[\![(\text{steq } s\ t)\theta\rho]\!]) = \phi(\mathcal{AQ}_\mathcal{R}[\![\text{true}]\!]) = \text{true}.$$

Hence $\mathcal{AH}_\mathcal{R} \models \exists X(\text{steq } s\ t = \text{true})$. ■

We define an S-indexed subclass $\mathcal{AD} = \{\mathcal{AD}^\sigma \mid \sigma \in S\} \subseteq \mathcal{AT}(\varnothing)$ as follows:

$$\begin{array}{rcl} \mathcal{AD}^o & = & \{ \text{ all ground data terms } \}, \\ \mathcal{AD}^{\sigma\to\tau} & = & \mathcal{AT}(\varnothing)^{\sigma\to\tau} \text{ for each } \sigma, \tau \in S. \end{array}$$

This class $\mathcal{AD}$ plays an intermediate role for witnesses between $\mathcal{AH}_\mathcal{R}$ and $\mathcal{AQ}_\mathcal{R}$.

Lemma 6.2
Let $\mathcal{R}$ be a program and $\exists X(\text{steq } s\ t = \text{true})$ a query.

$$\begin{array}{rl} & \mathcal{AH}_\mathcal{R} \models \exists X(\text{steq } s\ t = \text{true}) \\ \Rightarrow & \mathcal{AQ}_\mathcal{R} \models \forall\varnothing((\text{steq } s\ t)\xi = \text{true}) \text{ for some } \xi : X \to \mathcal{AD}. \end{array}$$

Proof Suppose δ is a witness of $\exists X(\text{steq}(s,t) = \text{true})$, i.e., a mapping $\delta : X \to \mathcal{AH}_\mathcal{R}$ which satisfies $\delta^\#(\text{steq}(s,t)) = \text{true}$. The assignment δ assigns for each $x \in X$ either a base type element or function type element. To classify these two, we use meta-variables x_i's for the variables assigning to the base type elements and y_i's for the variables assigning to the function type elements. Namely, we assume the following:

$$\begin{array}{rcl} X & = & \{x_1, \ldots, x_m, y_1, \ldots, y_n\}, \\ \delta & = & \{x_1 \mapsto h_1, \ldots, x_m \mapsto h_m, y_1 \mapsto f_1, \ldots, y_n \mapsto f_n\} \end{array}$$

where h_i's are the base type elements and f_j's are the function type elements. We fix these f_i's. Define a function ans : $\mathrm{H}^{m+n} \to \mathrm{H}$ such that

$$\begin{aligned} \mathrm{ans}(h_1, \ldots, h_m, f_1, \ldots, f_n) &= \mu^{\#}(\mathrm{steq}\ s\ t) \\ \text{where } \mu(x_i) &= h_i \text{ for } i = 1, \ldots, m, \\ \mu(y_i) &= d_i \text{ for } i = 1, \ldots, n. \end{aligned}$$

The function "ans" returns the value "true" if the arguments are witnesses of the query $\exists X(\mathrm{steq}\ s\ t)$ in $\mathcal{AH}_{\mathcal{R}}$. Note that the function "ans" is continuous. Then,

$$\begin{aligned} & \delta^{\#}(\mathrm{steq}\ s\ t) \\ = \ & \mathrm{ans}(h_1, \ldots, h_m, f_1, \ldots, f_n) \\ = \ & \mathrm{ans}(\bigsqcup\{z_1 \mid \mathrm{compact}\ z_1 \sqsubseteq h_1\}, \ldots, \bigsqcup\{z_1 \mid \mathrm{compact}\ z_1 \sqsubseteq h_m\}, f_1, \ldots, f_n) \\ = \ & \bigsqcup\{\mathrm{ans}(z_1, \ldots, z_m, f_1, \ldots, f_n) \mid \mathrm{compact} z_i \sqsubseteq h_i \text{ for each } i = 1, \ldots, m\} \\ & \qquad \text{(by the continuity of ans)} \\ = \ & \mathrm{true} \ \text{ (by assumption)} \end{aligned}$$

We fix z_i's determined above. Then we see that there exist compact elements $\hat{z_1}, \ldots, \hat{z_m} \in \mathrm{H}$ such that

$$\mathrm{ans}(\hat{z_1}, \ldots, \hat{z_m}, f_1, \ldots, f_n) = \mathrm{true}.$$

Taking arbitrary elements $d_1, \ldots d_n \in \mathcal{AD}^o$ that satisfy $d_1 \sqsupseteq \hat{z_1}, \ldots, d_n \sqsupseteq \hat{z_m}$, we obtain

$$\mathrm{ans}(d_1, \ldots, d_m, f_1, \ldots, f_n) = \mathrm{true}.$$

By the construction of $\mathcal{AH}_{\mathcal{R}}$, for each f_i, there exists $u_i \in \mathcal{AT}(\varnothing)$ such that $\mathcal{AH}_{\mathcal{R}}[\![u_i]\!] = f_i$. Hence letting $\xi = \{x_1 \mapsto d_1, \ldots, x_m \mapsto d_m, y_1 \mapsto u_1, \ldots, y_m \mapsto u_m\}$, we have

$$\mathcal{AQ}_{\mathcal{R}} \models \forall\varnothing\ ((\mathrm{steq}\ s\ t)\xi = \mathrm{true}).$$

■

Theorem 6.3 [**Completeness of Narrowing for $\mathcal{AH}_{\mathcal{R}}$**]
Let $\mathcal{R}$ be a program and $\exists X(\mathrm{steq}\ s\ t = \mathrm{true})$ a query. If

$$\mathcal{AH}_{\mathcal{R}} \models \exists X(\mathrm{steq}\ s\ t = \mathrm{true})$$

holds with a witness $\delta : X \to \mathcal{AH}_{\mathcal{R}}$ then there exists

- a narrowing derivation $(\mathrm{steq}\ s\ t) \leadsto^*_\theta \mathrm{true}$ with a answer substitution $\theta : X \to \mathcal{AT}(Y)$,
- a mapping $\phi : \mathcal{AH}_{\mathcal{R}} \to \mathcal{AD}$ which depends on δ and θ, and

- a substitution $\rho : Y \to \mathcal{AD}$ which depends on θ and ϕ

such that

$$\rho^{\#} \circ \theta = \phi \circ \delta.$$

Proof We will show the following diagram commutes:

$$\begin{array}{ccccc} X & = & X & = & X \\ {\scriptstyle\delta}\downarrow & & \downarrow{\scriptstyle\xi} & & \downarrow{\scriptstyle\theta} \\ \mathcal{AH}_{\mathcal{R}} & \xrightarrow{\phi} & \mathcal{AD} & \xleftarrow{\rho^{\#}} & \mathcal{AT}(Y) \end{array}$$

where ξ is a mapping determined in the following.

Suppose δ is a witness of $\exists X(\text{steq}(s,t) = \text{true})$, i.e., $\delta : X \to \mathcal{AH}_{\mathcal{R}}$ such that $\delta^{\#}(\text{steq}(s,t)) = \text{true}$. By Lemma 6.2, there exists an assignment

$$\xi : X \to \mathcal{AD} \text{ such that } \xi^{\#}(\text{steq } s\ t) = \text{true}.$$

By the Axiom of Choice, there exists a function Ψ such that

$$\Psi(\delta) = \xi.$$

Define a function $\phi : \mathcal{AH}_{\mathcal{R}} \to \mathcal{AD}$ such that

$$\phi(h) \stackrel{def}{=} \begin{cases} \Psi(\delta)(x) & \text{if there exists } x \in X \text{ such that } \delta(x) = h \\ \bot & \text{otherwise.} \end{cases}$$

Then clearly $\xi = \phi \circ \delta$. Since $(\text{steq } s\ t)\xi \to^*_{\mathcal{R}} \text{true}$, by the completeness of narrowing for rewriting, there exists a narrowing derivation $(\text{steq } s\ t) \leadsto^*_{\theta} \text{true}$ with an answer substitution $\theta : X \to \mathcal{AT}(Y)$ such that $\theta \preccurlyeq \xi$, i.e., there exists a substitution

$$\rho : Y \to \mathcal{AD} \text{ such that } \rho^{\#} \circ \theta = \xi.$$

Hence $\rho^{\#} \circ \theta = \phi \circ \delta$. ■

7. Conclusion

We have presented two models of a higher-order functional-logic language Ev, which are the quotient applicative Herbrand model and the minimal applicative Herbrand model and showed the soundness and completeness of narrowing with respect to both models in algebraic framework.

References

[1] J. A. Bergstra and J. W. Klop. Conditional rewrite rules: Confluence and termination. *Journal of Computer and System Sciences*, 32(3):323–362, 1986.

[2] E. Giovannetti, G. Levi, C. Moiso, and C. Palamidessi. Kernel-LEAF: A logic plus functional language. *Journal of Computer and System Sciences*, 42(2):139–185, 1991.

[3] J. Goguen and J. Meseguer. Models and equality for logical programming. *Lecture Notes in Computer Science 250*, pages 1–22, 1987.

[4] J. C. González-Moreno, M.T.Hortalá-González, and M. Rodríguez-Artalejo. Denotational versus declarative semantics for functional programming. *Lecture Notes in Computer Science 626*, pages 134–148, 1992.

[5] J. C. González-Moreno, M.T.Hortalá-González, and M. Rodríguez-Artalejo. On the completeness of narrowing as the operational semantics of functional logic programming. *Lecture Notes in Computer Science 702*, pages 216–230, 1992.

[6] M. Hamana, T. Nishioka, K. Nakahara, A. Middeldorp, and T. Ida. A design and implementation of a functional-logic language based on applicative term rewriting systems. *Transactions of Information Processing Society of Japan*, 36(8):1897–1905, 1995.

[7] Makoto Hamana. Semantics of a functional-logic language. Master's thesis, University of Tsukuba, 1995.

[8] M. Hanus. Compiling logic programs with equality. In *Proceedings of the 2nd Workshop on Programming Language Implementation and Logic Programming, Lecture Notes in Computer Science 456*, pages 387–401, 1990.

[9] T. Ida and S. Okui. Outside-in conditional narrowing. *IEICE Transactions on Information and Systems*, E77-D(6), 1994.

[10] S. Kaplan. Conditional rewrite rules. *Theoretical Computer Science*, 33(2):175–193, 1984.

[11] G. Levi, C. Palamidessi, P. G. Bosco, E. Giovannetti, and C. Moiso. A complete semantics characterization of K-LEAF, a logic language with partial functions. In *Proceedings 1987 Symposium on Logic Programming*, pages 318–327, Rockville, MD, 1987. IEEE Computer Society Press.

[12] K. Meinke. Universal algebra in higher types. *Theoretical Complete Science*, 100:385–417, 1992.

[13] A. Middeldorp and E. Hamoen. Completeness results for basic narrowing. *Applicable Algebra in Engineering, Communication and Computing*, 5(3/4):213–253, 1994.

[14] J. J. Moreno-Navarro and M. Rodríguez-Artalejo. Logic programming with functions and predicates: The language BABEL. *Journal of Logic Programming*, 12:191–223, 1992.

[15] K. Nakahara, A. Middeldorp, and T. Ida. A complete narrowing calculus for higher-order functional logic programming. In *Proceedings of Seventh International Conference on Programming Languages: Implementations, Logics, and Programming 95 (PLILP'95), Lecture Notes in Computer Science 982*, pages 97–114, 1995.