関数型プログラムの等号形式を含む論理プログラムの意味論

山崎 進　　広 朋之

岡山大学

FP 関数の項間の等号形式を含む論理プログラムについて論ずる。本報告の扱う言語の構文は Giovannetti らの LEAF に似ているが、意味論を等式の評価を含むデータフローを反映した代数的操作系の不動点で与えている。これにより、論理プログラムと関数プログラムの統合に関するデータフローの役割を示す。

Semantics for Logic Program with FP Functions Equality Based on Dataflow

Susumu Yamasaki and Tomoyuki Hiro

Department of Information Technology,
Okayama University, Okayama, Japan

This paper is concerned with FPLOG– a logic language with equality between terms involving FP functions. Its syntactic aspect follows that of LEAF (Giovannetti et al., 1991). On the basis of dataflow computing for FP functions as well as definite clause sets, the semantics for the language FPLOG is defined as denoting a substitution manipulator formed by an equation set with evaluations of equalities without term rewriting rules. So far it is shown the dataflow approach might be means of integrating logic and functional languages from semantic point.

1. Introduction

The integrated languages of logic, equation and function have been intensively studied. Among them, Prolog-with-Equality, Eqlog, Tablog, LEAF and Logic Programming with Equations might be thought of as the logic language with computing mechanism for equation and/or function (DeGroot et al., 1986; van Emden et al., 1987). Following the characteristic of these languages, exploiting plenty of expressiveness of FP functions, and extracting dataflow as substitution manipulation involved in a logic program, we define a language FPLOG, a logic language with FP functions equality and its semantics based on dataflow.

In the logic language, the substitution is of significance, as it is directly concerned with deductions as computing. There were approaches to deal with answer substitutions over a sequence domain, caused by deductions (Fitting, 1985; Debray and Mishra, 1987; Baudinet, 1988). Based on virtual bottom-up inference, there is another approach to treat sequences of substitutions from the point to interpret a logic program as denoting a dataflow, or to transform a logic program into a dataflow realizing its denotation (Yamasaki, 1990). Adopting the latter approach with algebraic manipulation on substitutions, this paper proposes a new treatment with semantics for a logic program based on dataflow. Intuitively each atom is represented by a substitution applied to a standardized atom. The standardized atom consists of a predicate symbol or an FP function symbol with equality, and individual variables without function constructor symbols. It is interpreted as denoting a sequence of substitutions to form atoms such that the sequence is associated with a functional representing and reflecting a bottom-up inference or atom generation. The functional is expressed only in terms of algebraic substitution manipulation. To get the functional, each definite clause might be regarded as a translator of substitution sequences to a substitution sequence, realizing a bottom-up inference in which the head is generated by assuming the substitutions (representing atoms) to be unifiable with the atoms in a body. The treatment is united with computing for equality.

In Section 2, the syntax of FPLOG is given, as well as the idempotent substitution set with a relation. In Section 3, the resolution deduction will be represented in terms of substitution manipulation, for the real atom form to be replaced. In Section 4, a functional over a sequence domain based on the idempotent substitution set is associated with a given logic program with FP functions equality. It is interpreted as dataflow to virtually realize the deductions for a logic program. The evaluation of equality will be arranged to be in solidarity with the deduction. In Section 5, the fixpoint semantics of the above functional is mentioned such that it is sound and complete for the deduction and the evaluation of equality if we permit the renaming of variables.

2. Logic Program with FP Functions Equality–FPLOG–

We deal with a logic program as a set of definite clauses in which FP functions might be exploited to form predicates with equality. Here we mean the functions in FP system (Backus, 1978) by FP functions. Its syntax is given by modifying that of LEAF (Giovannetti et al., 1991). Because the top level constructor is always arranged (or programmed) to be an FP function.

A term is recursively defined: (i) a variable is a term, and (ii) $f(t_1, \ldots, t_k)$ $(k \geq 0)$ is a term if f is a k-place function symbol and t_j are terms.

An FP-term is either (i) a term, or (ii) an expression $F(u_1, \ldots, u_n)$, where F is an FP function and u_i are FP-terms.

An atom is either an FP-atom as defined below, or an expression $P(t_1, \ldots, t_m)$, where P is a predicate symbol and t_i are terms.

An FP-atom is an expression $F(u_1, \ldots, u_n) = u$ for an FP function F, where $F(u_1, \ldots, u_n)$ and u are FP-terms. By the top level FP function of such an FP atom, the function F is meant.

A definite clause is a form $A \leftarrow B_1 \ldots B_n$ $(n \geq 0)$, where A, B_1, …, and B_n are atoms.

A logic program is a set of definite clauses, where for each FP-atom in a body of any clause there exists an FP-atom in the head of a clause such that the syntactic structure of the top level FP function is the same.

FPLOG is a language consisting of the logic programs in the above sense as well as their semantics.

Example 2.1. A simple example for an FPLOG program is

$$\left\{\begin{array}{rl} \mathit{Sum}(x,0) \leftarrow & \mathbf{length}(x) = 0, \\ \mathit{Sum}(x,y) \leftarrow & \mathit{Sum}(u,v), \mathbf{tl}(x) = u, \\ & +_{FP}(\mathbf{1}(x), v) = y, \\ \mathbf{length}(x_1) = y_1 \leftarrow & , \\ \mathbf{1}(x_2) = y_2 \leftarrow & , \\ \mathbf{tl}(x_3) = y_3 \leftarrow & , \\ +_{FP}(x_4, x_5) = y_4 \leftarrow & , \end{array}\right.$$

where *Sum* stands for a predicate symbol, 0 for a 0-place function symbol, x, y, u, v, x_1, y_1, x_2, y_2,

x_3, y_3, x_4, x_5, y_4 for variables, and **length**, **1** (selector), **tl** (tail), $+_{FP}$ (add) are FP functions to be evaluated. The program is intended to express the summation of the values of elements in a list.

As stated in Introduction, the semantics for FPLOG will be defined such that each logic program, which might involve FP functions, is regarded as a substitution manipulator based on dataflow. It comes up with us, mainly because the FP function can be easily realized by a dataflow. To establish the semantics, we investigate properties of substitutions which are usually effects in resolution deductions, but might be means to dismiss real atom forms.

For example, $A \leftarrow B_1 \ldots B_n$ might be represented by the tuple $(\varphi, \theta_1, \ldots, \theta_n)$ for $\{ P(\bar{x}), Q_1(\bar{x}_1), \ldots, Q_n(\bar{x}_n) \}$ such that $A = P(\bar{x})\varphi$, and $Q_i(\bar{x}_i)\ \theta_i = B_i$, $1 \leq i \leq n$, where $\bar{x}$ and $\bar{x}_i$ are newly provided tuples of variables, and φ, $\theta_1, \ldots,$ and θ_n are substitutions. Note new variables are adopted for the simplicity of treatments with substitution manipulations.

A substitution is a function from the set Var of variables to the set $Term$ of terms. $Dom(\theta)$, for a substitution θ, means the domain of θ, that is, $\{x \mid \theta(x) \neq x \text{ for } x \in Var\}$. If $Dom(\theta) = \{x_1, \ldots, x_n\}$, θ is represented by $\{ x_1 \mid \theta(x_1), \ldots, x_n \mid \theta(x_n) \}$. If $Dom(\theta)$ is empty, that is, $\theta(x) = x$ for any $x \in Var$, θ is especially denoted by ε. A substitution is said a permutation if it is a bijection from Var to Var.

The composition of substitutions φ, θ, denoted by $\varphi\theta$, is a substitution which means the application of φ to the result after applying θ.

We take a class of idempotent substitutions, which have favourable properties as in Eder (1985) and Palamidessi (1990).

Definition 2.2. A substitution θ is said idempotent if $\theta\theta = \theta$. *Sub* means the set of idempotent substitutions. For $\theta, \varphi \in Sub$, $\theta \preceq \varphi$ means there exists a substitution ρ such that $\varphi = \rho\theta$. $\theta \sim \varphi$ for $\theta, \varphi \in Sub$ iff $\theta \preceq \varphi$ and $\varphi \preceq \theta$.

In Eder (1985), it is shown if $\theta \sim \varphi$ then there exist permutations α, β such that $\alpha\theta = \varphi$ and $\beta\varphi = \theta$.

3. Substitution Manipulations for Deduction

In this section, in terms of substitutions we represent the resolution deduction, which is thought of as computing mechanism for the logic program except the evaluation process of FP functions.

For an nonempty subset At of atoms, we define

$$mgu(At) = \{\sigma \in Sub \mid \sigma \text{ is a most general unifier of any two atoms } A_1 \text{ and } A_2 \in At\}.$$

When At is empty, we regard $mgu(At)$ as $\{\varepsilon\}$. For the lemma to state there exists an idempotent unifier which is most general for a unifiable set, see Eder (1985).

The following definition is intended to express simultaneous unification.

Definition 3.1. *consis*: $Sub \times Sub \rightarrow 2^{Sub}$ and *comb*: $Sub \times Sub \rightarrow 2^{Sub}$ are defined as follows.

$$\begin{aligned} consis(\theta_1, \theta_2) &= \{\rho \in Sub \mid \exists \sigma_1, \sigma_2 : \sigma_1\theta_1 = \sigma_2\theta_2 = \rho\}, \\ comb(\theta_1, \theta_2) &= \{\rho \in Sub \mid \rho \text{ is in } consis(\theta_1, \theta_2) \text{ and } \rho \text{ is most general}\} \end{aligned}$$

$comb(\theta_1, \theta_2)$ is denoted by $\theta_1 + \theta_2$.

As an example, let $\theta_1 = \{x \mid f(y)\}$ and $\theta_2 = \{y \mid g(z)\}$. Then $\{x \mid f(g(z)), y \mid g(z)\} \in \theta_1 + \theta_2$.

As easily seen, $(\theta_1 + \theta_2) + \theta_3 = \theta_1 + (\theta_2 + \theta_3)$, which will be represented as $\theta_1 + \theta_2 + \theta_3$. We define $\theta_1 + \ldots + \theta_n = \{\varepsilon\}$ if $n = 0$.

It is easy to see:

Lemma 3.2. For given $\theta_1, \ldots, \theta_n \in Sub$, $\theta_1 + \ldots + \theta_n /\!\sim$ consists of just one equivalence class. Also $\theta_1 + \ldots \theta_n = \theta'_1 + \ldots + \theta'_n$ if $\theta_i \sim \theta'_i$, $1 \leq i \leq n$.

Note the *comb* function is closely related with $\dagger$ (least upper bound) of two equivalence classes of idempotent substitutions under the partial order $\leq$, which is induced on $Sub/\sim$ from $\preceq$ (Palamidessi, 1990). According to Eder (1985) or Palamidessi (1990), we can have a method to get an idempotent substitution in $\theta_1 + \ldots + \theta_n$ for given $\theta_1, \ldots, \theta_n$.

$+$: $Sub \times Sub \rightarrow 2^{Sub}$ might be extended to act on $2^{Sub} \times 2^{Sub}$ by:

$$\Theta_1 + \Theta_2 = \cup_{\theta_1 \in \Theta_1, \theta_2 \in \Theta_2}\ \theta_1 + \theta_2.$$

It follows $(\Theta_1 + \Theta_2) + \Theta_3 = \Theta_1 + (\Theta_2 + \Theta_3)$, which will be represented by $\Theta_1 + \Theta_2 + \Theta_3$. We regard $\Theta_1 + \ldots + \Theta_n = \{\varepsilon\}$ if $n = 0$.

The primary aim to take the *comb* function is to formulate the manipulation for the unit resolution deduction just only in terms of idempotent substitutions.

A unit (resolution) deduction from a set S of clauses is a sequence of definite clauses $G_1, \ldots, G_n$, where each G_h is either in S or inferred by unit resolution from some G_i and G_j $(i, j \leq h)$.

Unit resolution is an inference to derive $C\theta \leftarrow D_1\theta \ldots D_{i-1}\theta\ D_{i+1}\theta \ldots D_k\theta$ from two clauses $C \leftarrow D_1 \ldots D_k$ and $E \leftarrow$, where $\theta \in mgu(\{D_i, E\})$. In the usual case, E is regarded as having no common variables with $C \leftarrow D_1 \ldots D_k$ by renaming of variables. So the domain of θ might be restricted to the set of variables occurring in $C \leftarrow D_1 \ldots D_k$. Then the restriction of the domain of a substitution is exploited.

To restrict the domain of the substitution to some appropriate set, we have the following definition.

For a substitution σ and a set of atoms $\{A_1, \ldots, A_m\}$ $(m \geq 1)$, a restriction of σ with respect to $\{A_1, \ldots, A_m\}$, that is, $[\sigma]_{\{A_1,\ldots,A_m\}}$: $Var \rightarrow Term$ is defined as follows.

$$[\sigma]_{\{A_1,\ldots,A_m\}}(x) = \begin{cases} \sigma(x) & \text{if } x \text{ occurs in either } \\ & \quad A_1, \ldots, \text{ or } A_m, \\ x & \text{otherwise,} \end{cases}$$

for any $x \in Var$.

Definition 3.3. Assume Θ is a set of substitutions, and $\{A_1, \ldots, A_m\}$ $(m \geq 1)$ a set of atoms. We define

$$[\Theta]_{\{A_1,\ldots,A_m\}} = \begin{cases} \{[\theta]_{\{A_1,\ldots,A_m\}} \mid \theta \in \Theta\} & \\ & \text{if } \Theta \text{ is nonempty,} \\ \text{empty} & \text{if } \Theta \text{ is empty.} \end{cases}$$

Note if $\sigma \in Sub$, the $[\sigma]_{\{A_1,\ldots,A_m\}} \in Sub$. It is easy to see the following:

Lemma 3.4. $[\sigma]_{\{A_1,\ldots,A_m\}} \preceq \sigma$. If $\varphi \sim \theta$, then $[\varphi]_{\{A_1,\ldots,A_m\}} \sim [\theta]_{\{A_1,\ldots,A_m\}}$.

To represent an atom in terms of an adequate substitution, we interpret each atom as a standardized form with attached substitution. That is, we intend that an atom $P(t_1, \ldots, t_m)$ is denoted by $\theta = [\theta]_{\{P(x_1,\ldots,x_m)\}}$ for $P(x_1, \ldots, x_m)$ $(x_1, \ldots, x_m$: variables) such that $P(x_1, \ldots, x_m)\ \theta = P(t_1, \ldots, t_m)$. An FP-atom $F(u_1, \ldots, u_n) = u$ is denoted by $\varphi = [\varphi]_{\{F(y_1,\ldots,y_n)=y\}}$ for $F(y_1, \ldots, y_n) = y$ $(y, y_1, \ldots, y_n$: variables) such that $F(y_1, \ldots, y_n)\varphi = F(u_1, \ldots, u_n)$ and $y\varphi = u$. An atom $P(x_1, \ldots, x_m)$ or $F(y_1, \ldots, y_n) = y$ is referred to as a standard form.

For an atom C, $St(C)$ means a standard atom of C such that $St(C)$ involves a unique tuple of variables different from others, and $St(C)\ \varphi = C$ for some substitution φ, on the assumption any two atoms with the same predicate symbol or the top level FP function have the same standard atom.

To come up with a substitution manipulation for deductions, we firstly has a basic lemma, which is exploited to represent a set of most general unifiers by means of '+'.

Lemma 3.5. Assume $\varphi_1 = [\varphi_1]_{\{Q(\bar{x})\}} \in Sub$ and $\varphi_2 = [\varphi_2]_{\{Q(\bar{x})\}} \in Sub$, for a standard atom $Q(\bar{x})$, whether it is an FP-atom or not. Then $mgu(\{Q(\bar{x})\varphi_1, Q(\bar{x})\varphi_2\}) = [\varphi_1 + \varphi_2]_{\{Q(\bar{x})\varphi_1, Q(\bar{x})\varphi_2\}}$.

Proof. Assume σ is a most general unifier of $Q(\bar{x})\varphi_1$ and $Q(\bar{x})\varphi_2$. Then $(Q(\bar{x})\varphi_1)\sigma = (Q(\bar{x})\varphi_2)\sigma$. It follows from $\varphi_1 = [\varphi_1]_{\{Q(\bar{x})\}}$ and $\varphi_2 = [\varphi_2]_{\{Q(\bar{x})\}}$ that $\sigma\varphi_1 = \sigma\varphi_2$. $Dom(\sigma)$ suggests $\sigma = [\sigma\varphi_1]_{\{Q(\bar{x})\varphi_1, Q(\bar{x})\varphi_2\}} = [\sigma\varphi_2]_{\{Q(\bar{x})\varphi_1, Q(\bar{x})\varphi_2\}}$. That is, $\sigma \in [\varphi_1 + \varphi_2]_{\{Q(\bar{x})\varphi_1, Q(\bar{x})\varphi_2\}}$. On the other hand, assume that $\sigma \in [\varphi_1 + \varphi_2]_{\{Q(\bar{x})\varphi_1, Q(\bar{x})\varphi_2\}}$. Then $\sigma = [\rho_1\varphi_1]_{\{Q(\bar{x})\varphi_1, Q(\bar{x})\varphi_2\}} = [\rho_2\varphi_2]_{\{Q(\bar{x})\varphi_1, Q(\bar{x})\varphi_2\}}$ for some ρ_1 and ρ_2 such that $\rho_1\varphi_1 = \rho_2\varphi_2 \in \varphi_1 + \varphi_2$. It follows $\sigma = [\rho_1]_{\{Q(\bar{x})\varphi_1, Q(\bar{x})\varphi_2\}} = [\rho_2]_{\{Q(\bar{x})\varphi_1, Q(\bar{x})\varphi_2\}}$ and $(Q(\bar{x})\varphi_1)\sigma = (Q(\bar{x})\varphi_2)\sigma$. That is, σ is a unifier of $Q(\bar{x})\varphi_1$ and $Q(\bar{x})\varphi_2$. σ should be most general in the set of unifiers, as $\rho_1\varphi_1 = \rho_2\varphi_2 \in \varphi_1 + \varphi_2$ and $\sigma = [\rho_1\varphi_1]_{\{Q(\bar{x})\varphi_1, Q(\bar{x})\varphi_2\}} = [\rho_2\varphi_2]_{\{Q(\bar{x})\varphi_1, Q(\bar{x})\varphi_2\}}$.

Following the above lemma, we have a lemma to paraphrase the (unit) deduction by means of '+'.

Lemma 3.6. $A' \leftarrow$ might be deduced by unit deduction from a definite clause set $\{A \leftarrow B_1 \ldots B_n, B'_1 \leftarrow, \ldots, B'_n \leftarrow\}$, where each B'_i has no common variable with $A \leftarrow B_1 \ldots B_n$ iff $A' = A\theta$, unless $A' \neq B'_i$ $(1 \leq i \leq n)$, for $\theta \in [\sigma_1 + \ldots + \sigma_n]_{\{A\}}$ such that $\sigma_i = [\sigma_{i0}]_{\{B_i\}}$ $(1 \leq i \leq n)$, where σ_{i0} is a most general unifier (mgu) σ_{i0} of B_i and B'_j for some B'_j.

Proof. Assume $\theta \in [\sigma_1 + \ldots + \sigma_n]_{\{A\}}$. Then $\theta = [\theta_0]_{\{A\}}$ for some $\theta_0 \in \sigma_1 + \ldots + \sigma_n$. It follows $\theta_0 = \rho_1\sigma_1 = \ldots = \rho_n\sigma_n$ for some $\rho_1, \ldots, \rho_n$. Since $B'_i\ \varphi_i = B_i\theta_0$ for some φ_i $(1 \leq i \leq n)$, $A\theta \leftarrow$ might be deduced from $\{A\theta_0 \leftarrow B_1\theta_0 \ldots B_n\theta_0, B'_1 \leftarrow, \ldots, B'_n \leftarrow\}$. By Lifting Lemma (See Chan et al., 1973), $A\theta' \leftarrow$ is derivable from the given definite clause set, for $\theta' = [\theta']_{\{A\}} \preceq \theta$. However, because of most generality of θ_0 such that $\theta_0 = \rho_1\sigma_1 = \ldots \rho_n\sigma_n$ for some $\rho_1, \ldots, \rho_n$, $\theta' = [\theta_0]_{\{A\}} = \theta$. On the other hand, assume $A' \leftarrow$ might be deduced from the given definite clause set. Then there exist θ, θ_0 such that θ_0 simultaneously unify B_i and B'_j $(1 \leq i \leq n)$, θ_0 is most general, $\theta = [\theta_0]_{\{A\}}$ and $A' = A\theta$. It follows there exist substitutions $\rho_1, \ldots, \rho_n$ such that $\theta_0 = \rho_1[\sigma_{10}]_{\{B_1\}} = \ldots = \rho_n[\sigma_{n0}]_{\{B_n\}}$. (Note B'_j has no common variables with B_i.) Finally $\theta_0 \in [\sigma_{10}]_{\{B_1\}} + \ldots + [\sigma_{n0}]_{\{B_n\}} = \sigma_1 + \ldots + \sigma_n$.

By means of Lemma 3.6, we have:

Theorem 3.7. Assume a definite clause set $\{Q(\bar{x})\varphi \leftarrow Q_1(\bar{x}_1)\theta_1 \ldots Q_m(\bar{x}_m)\theta_m,\ Q_1(\bar{x}_1)\psi_1 \leftarrow,\ \ldots,\ Q_m(\bar{x}_m)\psi_m \leftarrow\ \}$, where each $Q_i(\bar{x}_i)\psi_i$ has no common variables with $Q(\bar{x})\ \varphi \leftarrow Q_1(\bar{x}_1)\ \theta_1 \ldots Q_m(\bar{x}_m)\ \theta_m$. Then $(Q(\bar{x})\varphi)\sigma \leftarrow (\sigma = [\sigma]_{\{Q(\bar{x})\varphi\}})$ is derivable by unit deduction from the definite clause set iff $\sigma \in [\varphi_1 + \ldots + \varphi_m]_{\{Q(\bar{x})\varphi\}}$ such that $\varphi_i \in [\theta_i + \psi_i]_{\{Q_i(\bar{x}_i)\theta_i\}}$, $1 \leq i \leq m$.

Proof. By Lemma 3.5, φ_i is a restriction of an *mgu* of $Q_i(\bar{x}_i)\ \theta_i$ and $Q_i(\bar{x}_i)\ \psi_i$, with respect to $Q_i(\bar{x}_i)\ \theta_i$ $(1 \leq i \leq m)$. It follows from Lemma 3.6 that $(Q(\bar{x})\varphi)\sigma \leftarrow (\sigma = [\sigma]_{\{Q(\bar{x})\varphi\}})$ is derivable iff $\sigma \in [\varphi_1 + \ldots + \varphi_m]_{\{Q(\bar{x})\varphi\}}$.

4. Denotation of FPLOG Based on Dataflow

We look for a method to give the denotation of FPLOG. Since an FP function might be realized by dataflow computing, the semantics for FPLOG is expected by extending the result in Yamasaki (1990), and by assuming the evaluation mechanism for the FP-atom with a substitution, which is executed based on dataflow by adopting the way as in Hankin et al. (1987) with the *first* and *next* functions in Ashcroft et al. (1976).

For an assumed logic program L:

$$(4.1)\left\{\begin{array}{lll} C_1 \leftarrow & D_{1,1} \ldots D_{1,n_1}, \\ \ldots & \ldots \\ C_k \leftarrow & D_{k,1} \ldots D_{k,n_k}, \end{array}\right.$$

we are supposed to obtain the denotation of the standard form $St(C_i)$ for C_i, $|\ St(C_i)\ |$ $(1 \leq i \leq k)$ over a sequence domain such that for each $p \in \omega$ (the set of natural numbers) $St(C_i)\ (|\ St(C_i)\ |\ (p)) \leftarrow$ is deducible from L or undefined, with atoms affected by substitutions. Note $|\ St(C_i)\ |$ is a sequence and the sequence is regarded as a partial function from the set of natural numbers to the set of idempotent substitutions. To denote time delay in a sequence, we exploit hiaton (denoted by τ) as in Park (1983). Then we take a sequence domain $(Sub \cup \{\tau\})^\infty$, which consists of all finite and infinite sequences from $Sub \cup \{\tau\}$, where $u \in (Sub \cup \{\tau\})^\infty$ is a partial function from ω to $Sub \cup \{\tau\}$ such that $u \in (Sub \cup \{\tau\})^\infty$ iff $u(p) \in Sub \cup \{\tau\}$ whenever $u(q) \in Sub \cup \{\tau\}$ and $p \leq q$.

Theorem 3.7 induces the recursive relation among the denotations of the standard forms.

Assume for $C_i \leftarrow D_{i,1} \ldots D_{i,n_i}$ that $C_i = St(C_i)\varphi_i$, $D_{i,j} = St(D_{i,j})\theta_{i,j}$ $(1 \leq i \leq k,\ 1 \leq j \leq n_i)$, and $|\ St(D_{i,j})\ |\ (q_j) = \psi_{i,j}$. Then we may define

$$\begin{aligned} &|\ St(C_i)\ |\ (p) \\ &\quad \in \{\sigma_i(p)\varphi_i \mid \sigma_i(p) \in [\varphi_{i,1} + \ldots + \varphi_{i,n_i}]_{\{St(C_i)\varphi_i\}}\}, \end{aligned}$$

where $\varphi_{i,j} \in [\theta_{i,j} + \psi_{i,j}]_{\{D_{i,j}\}}$, $1 \leq j \leq n_i$, when the $(p+1)$–th item (an output) in a sequence $|\ St(C_i)\ |$ is supposed to be decided by the inputs, that is, the (q_1+1)–the item of $|\ St(D_{i,1})\ |$, $\ldots$, and the $(q_{n_i}+1)$–th item of $|\ St(D_{i,n_i})\ |$.

Since $[\ \varphi_{i,1} + \ldots + \varphi_{i,n_i}]_{\{St(C_i)\varphi_i\}}$ consists of just one equivalence class of idempotent substitutions by Lemmas 3.2 and 3.4, we might select an idempotent substitution as the value of $|\ St(C_i)\ |\ (p)$ modulo $\sim$, for the above relation to be an equation.

To correlate the position of an output sequence with the positions of input sequences by one-to-one correspondence, we take a bijection I_m: $\omega \rightarrow \omega^m$, and a projection $J_{m,i}$: $\omega^m \rightarrow \omega$ such that $J_{m,i}(q_1, \ldots, q_m) = q_i$. The composition $J_{m,i} \circ I_m$ is denoted by $I_{m,i}$, that is, $I_{m,i}(q) = J_{m,i}(I_m(q))$.

Next observe $St(C_i)(|\ St(C_i)\ |\ (p)) \leftarrow$ might be interpreted as unifiable with some atom in a body of another definite clause, if $St(C_i)(|\ St(C_i)\ |\ (p))$ is defined as a real atom.

If C_i is an FP atom, we should investigate the validity of the equality.

Definition 4.1. We introduce *Eval* : $SATOM \times (Sub \cup \{\tau\}) \rightarrow Sub \cup \{\tau\}$ such that

$$\begin{aligned} &Eval(F(x_1, \ldots, x_m) = x, \varphi) \\ &\quad = \left\{\begin{array}{ll} \varphi & \text{if } F(x_1, \ldots, x_m)\varphi = x\varphi, \\ \tau & \text{otherwise,} \end{array}\right. \end{aligned}$$

where *SATOM* means the set of standard FP-atoms, and τ is a special symbol not in *Sub*.

Note $F(x_1, \ldots, x_m)\varphi = x\varphi$ holds if both sides are equal either syntactically or in value. Also note the above *Eval* might be executed by dataflow (as in Hankin et al. with the *first* and *next* functions in Ashcroft et al., for example) and τ is interpreted as time delay or hiaton as in Park (1983).

Definition 4.2. For a standard FP-atom A, we define $form(|\ A\ |)(p) = Eval(A, |\ A\ |\ (p))$, where $|\ A\ |$ means the denotation of the FP-atom A. For a standard non-FP-atom B, we set $form(|\ B\ |) = |\ B\ |$.

Therefore if A is a standard FP-atom then $form(|\ A\ |)$ denotes a sequence in which each item is obtained by evaluating the equality of the corresponding item in $|\ A\ |$.

By means of the above observation and intention, we have an equation set of sequence variables for a logic program (4.1) in FPLOG, where the equation set might be interpreted as denoting a dataflow.

Firstly the logic program L is transformed to:

$$(4.2)\left\{\begin{array}{lll} St(C_1)\varphi_1 & \leftarrow & St(D_{1,1})\theta_{1,1}\ldots St(D_{1,n_1})\theta_{1,n_1} \\ \ldots & \ldots & \\ St(C_k)\varphi_k & \leftarrow & St(D_{k,1})\theta_{k,1}\ldots St(D_{k,n_k})\theta_{k,n_k} \end{array}\right.$$

Now we set

$$(4.3)\left\{\begin{array}{lll} Pred(L) & = & \{P_1,\ldots,P_h\}, \\ FP(L) & = & \{F_1,\ldots,F_l\}, \end{array}\right.$$

where $Pred(L)$ stands for the set of all predicate symbols occurring in L, and $FP(L)$ for the set of all syntactic aspects of FP functions occurring in L as the top level.

To the denotation of $St(C_1)$, ..., $St(C_k)$, we assign $form(u_1)$, ..., $form(u_k)$ by using sequence variables in $(Sub \cup \{\tau\})^\infty$, respectively. Some of them might be merged as a denotation of an atom involving the same predicate symbol in $Pred(L)$ or the same FP function in $FP(L)$. Classifying the predicate symbols and the top level FP functions, $v_1, \ldots, v_{h+l}$ are assigned to the denotations of standard atoms.

Now we prepare for a device to gather more than two sequences for the same standard atom. It might be the case that $St(C_i) = St(C_j)$ for $i \neq j$. A fair merge is most adequate. For time delay τ, it will be guaranteed to be continuous (See Park, 1983).

In order to use the fair merge, we put for the j−th predicate symbol or top level FP function

$$\begin{array}{lll} (4.4) & & \\ Stand(j) & = & \{i \mid \ C_i \leftarrow D_{i,1}\ldots D_{i,n_i} \in L \text{ and} \\ & & \quad St(C_i) \text{ involves } P_j \text{ or } F_j\} \\ & = & \{j_1,\ldots,j_{NO(j)}\}, \end{array}$$

where $NO(j)$ is the cardinal number of $Stand(j)$.

For $1 \leq j \leq h+l$, we have

$$(4.5)\ v_j = \ fairmerge^{NO(j)}(form(u_{j_1}), \ldots, form(u_{j_{NO(j)}})),$$

where $fairmerge^{NO(j)}$ is a function to provide an output sequence by merging $NO(j)$ input sequences without neglecting any part of any input.

Note $form(u_{j_i})$ will denote an infinite sequence by means of τ.

By means of Theorem 3.7 as well as (4.5) for (4.2), we might have, for $1 \leq i \leq k$,

$$\begin{array}{ll} (4.6) & \\ \multicolumn{2}{l}{u_i(p) \in \{\sigma_i(p)\varphi_i \mid \sigma_i(p) \in \Sigma_i(p)\} \text{ for}} \\ \multicolumn{2}{l}{\ \Sigma_i(p) = [\ [\theta_{i,1} + v_{i_1}(I_{n_i,1}(p))]_{\{D_{i,1}\}} + \ldots +} \\ \multicolumn{2}{l}{\qquad [\theta_{i,n_i} + v_{i_{n_i}}(I_{n_i,n_i}(p))]_{\{D_{i,n_i}\}}]_{\{St(C_i)\varphi_i\}}} \\ & \qquad \text{if } \Sigma_i(p) \text{ is defined and nonempty} \\ u_i(p) = \ \tau & \quad \text{otherwise,} \end{array}$$

where v_{i_j} is the denotation of $St(D_{i,j})$.

For the reason each $form(u_i)$ is thought of as an input by means of some v_j, and renaming of variables in a substitution is necessary for it to be another input, as well as for the reason we intend to have an equation from (4.6), (4.6) is revised:

For $1 \leq i \leq k$,

$$\begin{array}{ll} (4.7) & \\ u_i(p) & = [\rho_i(p)\sigma_i(p)\varphi_i]_{\{St(C_i)\}} \\ & \quad \text{for } \sigma_i(p) \in \ [\theta_{i,1} + v_{i_1}(I_{n_i,1}(p))]_{\{D_{i,1}\}} + \ldots + \\ & \qquad\qquad [\theta_{i,n_i} + v_{i_{n_i}}(I_{n_i,n_i}(p))]_{\{D_{i,n_i}\}} \\ & \quad \text{if } [\ldots]_{\{D_{i,1}\}} + \ldots + [\ldots]_{\{D_{i,n_i}\}} \text{ is defined and} \\ & \quad \text{nonempty,} \\ u_i(p) & = \tau \qquad \text{otherwise,} \end{array}$$

where $\rho_i(p)$ is a permutation such that $[\sigma_i(p)]_{\{St(C_i)\varphi_i\}} \sim [\rho_i(p)\sigma_i(p)]_{\{St(C_i)\varphi_i\}}$ and some term (involving no variable in L nor involving variables in standard atoms for L) is assigned to every variable in $St(C_i)\varphi_i$.

(4.7) as well as (4.5) is an equation set for a logic program L (in FPLOG). It is interpreted as realizing dataflow for L, in its operational form over sequence variables as well as by means of the function $form$.

5. Semantics for FPLOG

Assume a logic program in the form (4.1) or (4.2). Then we have constructed (4.7) as well as (4.5), based on Theorem 3.7. (4.5) and (4.7) might state the substitution generation (caused by deductions and evaluations of equalities described by FP functions), which virtually represent the atom generation by dismissing its real form.

(4.5) and (4.7) are described by the fixpoint semantics, which is concerned with the original logic program.

Firstly we need a partial order on $(Sub \cup \{\tau\})^\infty$.

Definition 5.1. A partial order $<$ on $Sub \cup \{\tau\}$ is induced by: $\tau < \theta$ and $\theta < \theta$ for any $\theta \in Sub$. A partial order $\sqsubset$ on $(Sub \cup \{\tau\})^\infty$ is defined by: $u \sqsubset v$ for $u, v \in (Sub \cup \{\tau\})^\infty$ iff $u(p) < v(p)$ for any $p \in \omega$ such that $u(p)$ is defined.

Lemma 5.2. Assume (4.5) and (4.7) for a logic program in (4.2). Then (4.5) and (4.7) define a continuous functional f_L: $((Sub \cup \{\tau\})^\infty)^{k+h+l} \rightarrow ((Sub \cup \{\tau\})^\infty)^{k+h+l}$ such that there is a least fixpoint of f_L.

Proof. In (4.7), $u_i(p) \in Sub \cup \{\tau\}$ for any $p \in \omega$, that is, u_i is infinite. $u_i(p)$ depends on $v_{i_1}(I_{n_i,1}(p)), \ldots, v_{i_{n_i}}(I_{n_i,n_i}(p))$ by one-to-one correspondence. Thus the item of each position in a sequence for u_i one-to-one corresponds to the tuples of items of some positions in sequences for $v_1, \ldots, v_{h+l}$. This means (4.7) is continuous. On the other hand, $form(u_i)$ provides an infinite sequence in pointwise accordance with u_i. For infinity, the $fairmerge$ function might be continuous (Park, 1983). Therefore (4.5) is continuous, depending on $u_1, \ldots, u_k$.

Notation 5.3. Assume (4.7) as well as (4.5) for a logic program L in (4.2). $(u_1^f, \ldots, u_k^f, v_1^f, \ldots, v_{h+l}^f)$ means the least fixpoint of the functional to be defined by (4.5) and (4.7).

By means of Theorem 4.7, and Definitions 4.1 and 4.2, we have:

Theorem 5.4. For a logic program L in (4.2) with (4.5) and (4.7), $P_l(\bar{x}_l)\psi_l \leftarrow$ is deducible from L if $\psi_l = v_l^f(p)$ for any $p \in \omega$ such that $v_l^f(p) \neq \tau$, and only if $\psi_l \sim v_l^f(q)$ for some q. Also $F_j(x_{j,1}, \ldots, x_{j,m_j})\psi_j = x_j\psi_j \leftarrow$ holds for L if $\psi_j = v_j^f(p)$ for any $p \in \omega$ such that $v_j^f(p) \neq \tau$, and only if $\psi_j \sim v_j^f(q)$ for some q.

Outline of Proof. Theorem 3.7 guarantees (4.7) as well as (4.5) is sound for the unit deductions. Since by Lemmas 3.2 and 3.4, + is an invariant operation for the operands of equivalent substitutions. In (4.5), the $fairmerge$ permits all the items in any sequence to be transferred, and the $form$ denotes the evaluations of equalities for FP-atoms. Thus (4.5) and (4.7) are complete for the unit deductions as long as renaming of variables might be permitted.

Therefore the semantics for FPLOG is provided by the least fixpoint of f_L defined by (4.5) and (4.7).

6. Concluding Remarks

The syntax of FPLOG, which is a logic program with FP functions equality is similar to that of LEAF. However, its semantics has characteristic aspect in that each predicate denotes a sequence with each item of which a real atom might be interpreted as being generated by deductions and evaluations of equalities, and the logic program denotes a functional as substitution manipulator over a sequence domain. Finally the whole logic program reflects and represents dataflow computing for deductions and evaluations of equalities. This primary conclusion is owing to the fair merge in Park (1983), compared with the dataflow languages (Kahn, 1976; Ashcroft et al., 1976).

The problem of how to establish the rewriting and evaluation system in which syntactically different FP functions might occur is still left for study from semantic points. However, we might have the conclusion that the operational semantics for FPLOG might be given by dataflow computing for the deductions as well as the evaluations of equalities. The dataflow computing for the deductions is an alternative to the finite deduction as in Apt et al. (1982) by means of elaborate, algebraic substitution manipulation with an interpretation of definite clauses as representing a functional over a sequence domain.

In this paper, the process of extracting functional from FPLOG means the construction of dataflow over sequences denoted by predicates, but not the construction of a function for each predicate as in Debray et al. (1989).

Acknowledgement

The first author is much indebted to the late Professor David M.R. Park and Dr.Stephen G.Matthews for their comments on dataflow computing during the author's visits to University of Warwick. This work is partially supported by the Japanese Ministry of Education, Science and Culture.

References

1. Apt,K.R. and van Emden,M.H., Contributions to the theory of logic programming, J.ACM 29, pp.841–864, 1982.

2. Ashcroft,E.A. and Wadge,W.W., Lucid–A formal system for writing and proving programs, SIAM J. Comput. 5, pp.336–354, 1976.

3. Backus,J., Can programming be liberated from the von Neumann style? A function style and its algebra of programs, C.ACM 21, pp.613–641, 1978.

4. Baudinet,M., Proving termination properties of PROLOG programs: A semantic approch, Proc. of 3rd Annual Symposium on Logic in Computer Science, pp.336–347, 1988.

5. Chang,C.L. and Lee,R.C.T., Symbolic Logic and Mechanical Theorem Proving, Academic Press, 1973.

6. Debray,S. and Mishra,P., Denotational and operational semantics for Prolog, in "Formal Description of Programming Concepts III (Wirsing,M., ed.), Proc. IFIP TC 2/WG 22, North-Holland, pp.245–270, 1987.

7. Debray,S. and Warren,D.S., Functional computations in logic programs, ACM Trans. on Programming Languages and Systems 11, 3, pp.451–481, 19891.

8. DeGroot,D. and Lindstrom,G. (eds.), Logic Programming: Functions, Relations and Equations, Prentice-Hall, 1986.

9. Eder,E., Properties of substitutions and unifiers, J. Symbolic Computation 1, pp.31–46, 1985.

10. van Emden,M.H. and Yukawa,K., Logic programming with equations, J. Logic Programming 4, pp.265–288, 1987.

11. Fitting,M., A deterministic Prolog fixpoint semantics, J. Logic Programming 2, pp.111–118, 1985.

12. Giovannetti,E. el al., Kernel-LEAF: A logic plus functional language, J. of Computer and System Sciences 42, pp.139–185, 1991.

13. Hankin,C.T.D. and Glaster,H., Applicative languages and dataflow, in "Functional Programming: Languages, Tools and Architectures" (Eisenbach,S., ed.), Ellis Horwood, pp.128–140, 1987.

14. Kahn,G., The semantics of a simple language for parallel programming, Proc. IFIP '74, pp.471–475, 1974.

15. Palamidessi,C., Algebraic properties of idempotent substitutions, Lecture Notes in Computer Science 443, pp.386–399, 1990.

16. Park,D., The "fairness" problem and nondeterministic computing networks, in "Foundations of Computer Science IV" (de Bakker,J.W. and van Leeuwen,J., eds.), Mathematisch Centrum, Amsterdam, pp.133–161, 1983.

17. Yamasaki.S., Recursion equation sets computing logic programs, Theoretical Computer Science 76, pp.309–322, 1990.