π-Calculus Semantics of Moded Flat GHC

Keiji Hirata

Technical Report: ISRL-95-3
August 1995

Information Science Research Laboratory
Basic Research Laboratories
Nippon Telegraph and Telephone Corporation

π-Calculus Semantics of Moded Flat GHC

Keiji Hirata
NTT Basic Research Laboratories
3-1, Morinosato Wakamiya, Atsugi-shi, Kanagawa, 243-01 Japan
hirata@nefertiti.brl.ntt.jp

Abstract

This paper presents a new operational semantics for moded Flat Guarded Horn Clauses (FGHC). π-calculus is a simple model of concurrent computation based upon the notion of naming; π-calculus agents concurrently exchange names as data via names as channels. Naming enables an agent to be encapsulated as a value and also can provide the execution view that independent states interact with one another. The semantics of π-calculus has been investigated intensively mainly from the algebraic point of view. Given an FGHC program, we can consider that the translated π-calculus statements represent the operational meaning of the source program. Such a π-calculus semantics has the following advantages: both processes and messages (terms) are represented in a uniform way, the semantics can specify nonlogical built-in predicates together with ordinary predicates and logical variables, and the various properties of π-calculus are well-understood theoretically. This paper introduces FGHC$^-$ (a subset of moded FGHC), presents the rules for translation from an FGHC$^-$ program to polyadic π-calculus statements, introduces the moded ground equality theory in order to define the unification in FGHC$^-$, and shows that the new semantics is correct with respect to the Ueda's conventional operational semantics of GHC based on a transition system.

1 Introduction

Guarded Horn Clauses (GHC) is a concurrent logic language based on parallelism in logic programming, whereas π-calculus is a model of concurrent computation based on the notion of naming. Both are simple yet powerful frameworks describing concurrent processes (referred to in π-calculus as *agents*). Although GHC and π-calculus have similar motivations and aims, they have quite different origins, concepts and formulations. Investigation of the link between GHC and π-calculus may thus provide new insights into concurrent computation.

A prominent feature of GHC (and FGHC) is that interprocess communication is implemented by unification, where logical variables play a crucial role. That is, it is provided through dataflow synchronization and broadcasting. A program consisting of clauses can be viewed as conditional rules for rewriting goals, and goal rewriting governs the unification of logical variables. Among those who have examined the relation of GHC and other similar paradigms, Saraswat [7] carefully discussed GHC in the concurrent constraint framework, Honda et al. [1] compared GHC features with those of the Actor model and

describes the implementation of GHC in actors, and Nyström [6] presented the execution model of FGHC described in the SCL language. So far there have been many studies on the semantics of GHC [8][5], but to examine the program's behavior and properties more precisely it would be convenient for the semantics to be constructed on an already well-studied domain or model. In this regard, we thought that π-calculus is noteworthy.

During the computation (reduction) of π-calculus, agents concurrently exchange names as data via names as channels, where a name is only an entity. Its semantics has been investigated intensively mainly from the algebraic point of view. Moreover, its connections to similar models have also been elaborated: e.g., λ-calculus [3], object-oriented languages [11], Prolog [2], and linear logic.

This paper presents a new operational semantics for moded FGHC based on π-calculus. Given an FGHC program, we can consider the translated π-calculus statements to represent the operational meaning of the source program. Such a π-calculus semantics has the following advantages: the various properties of π-calculus are well-understood theoretically, the semantics can specify nonlogical built-in predicates together with ordinary predicates and logical variables, and both processes and messages (terms) are represented in a uniform way.

This paper first introduces FGHC$^-$, which is a subset of moded FGHC, and then presents the rules for translation from an FGHC$^-$ program to polyadic π-calculus statements. Then, it introduces the moded ground equality theory in order to prescribe the unification in FGHC$^-$. Lastly, the translation is shown to be correct with respect to Ueda's conventional operational semantics of GHC based on a transition system. The new operational semantics based on π-calculus can be used for specifying new built-in predicates and provides a theoretical foundation for transforming programs to make execution more efficient.

2 Preliminaries

2.1 Moded FGHC

This section briefly reviews moded FGHC [10], which is FGHC with mode information. In moded FGHC every occurrence of a variable is associated with a mode, either *in* or *out*. A logical variable is considered a one-to-n ($n \geq 0$) communication channel that connects its occurrences. Roughly, occurrences moded *in* and *out* correspond to inlets and outlets of information. All variables in a well-moded program are guaranteed to be used for one-to-n ($n \geq 0$) communication; that is, for each variable there is one *out* occurrence and there are n *in* occurrences.

Some advantages of well-moded FGHC [10] can be stated informally as follows:

- Since each variable has one writer, active unification never fails.
- Given a well-moding of a program and a goal, if the goal is reduced by one step the reduced goal and the program have the same well-moding.
- Given a well-moding of a program and a goal, if the goal has been reduced to an empty set of goals, all the variables in the goal are mapped to ground terms.

The source language considered here is a subset of moded FGHC. It is denoted here as FGHC$^-$ and has the following syntax (in a BNF-like style):

$$
\begin{aligned}
Goal ::= &\ :- \ b_1, \cdots, b_n \\
Program ::= &\ c_1 \cdots c_n \\
c_i ::= &\ u(\vec{A}^{\mu}) :- g \mid b_1, \cdots, b_n \\
g ::= &\ true \mid A^i_j = f(\vec{D}^{\mu}) \mid A^i_j = A^i_k \mid g_builtin(\vec{E}^i) \\
b_i ::= &\ true \mid u(\vec{R}^{\mu}) \mid b_builtin(\vec{R}^{\mu}) \mid Y^o_j = f(\vec{R}^{\mu}) \mid Y^o_j = X^i_k \\
&\ A^i_j, A^i_k \in \vec{A}^{\mu},\ var(\vec{A}) \cup var(\vec{D}) \text{ are distinct},\ \vec{E} \subseteq \vec{A}
\end{aligned}
$$

Let $Pred$, Fun, and Var be the disjoint sets of predicate, function, and variable symbols; $u \in Pred$ and $f \in Fun$. Following convention, capital letters stand for logical variables. Every occurrence of a variable is associated with a mode, either *in* or *out*, and is written V^i or V^o. Let $\vec{V}^{\mu}$ be $V_1^{\mu_1}, \cdots, V_n^{\mu_n}$, μ_k be either *in* or *out*, and $var(\vec{V})$ be the set of variables in $\vec{V}$. The term $u(\vec{R}^{\mu})$ represents a user-defined predicate, and $g_builtin(\vec{E}^i)$ and $b_builtin(\vec{R})$ respectively represent a guard built-in predicate and a body built-in predicate.

FGHC$^-$ has the following properties: A goal and a program are well-moded. At the top level of arguments of the predicate and the function symbols, only variables occur. Function symbols occur only as the arguments of '=' both in the guard and the body. It is obvious that FGHC$^-$ has almost the same descriptive power as FGHC even though FGHC$^-$ is syntactically restricted.

Let a communication variable (*c-var*) be defined as a variable with both *in* and *out* modes occurring simultaneously in the body. A c-var may occur at most once either in the head (*out*) or the guard (*in*); there can be n (≥ 1) occurrences moded *in* in the body. Intuitively, in FGHC$^-$ the occurrence of a c-var implies that a variable which is singly assigned to and possibly multiply referred to by other processes is allocated at that moment. To that end, the c-var occurring in the body and moded *out* is written, for example, as Y_j^c and $\vec{Z}^c$. We do not care of whether or not the variables moded *in* are c-vars.

2.2 Polyadic π-Calculus

π-calculus agents (processes) exchange data through communication channels concurrently [4]. The data and the communication channels are represented by names, which are only entities in π-calculus. Polyadic π-calculus (PPC) has the syntax defined as follows:

$$
P ::= \textstyle\sum_{i \in I} \pi_i.P_i \ \Big|\ P|Q \ \Big|\ (\nu x_1 \cdots x_n)P \ \Big|\ \mathcal{A}[x_1 \cdots x_n] \ \Big|\ \mathbf{0}
$$

The term π_i is called a prefix and has two basic forms: $\overline{x}[y_1, \cdots, y_n]$ (polyadic output) and $x([y_1, \cdots, y_n])$ (polyadic input). $\overline{x}[y_1, \cdots, y_n].P$ atomically sends the tuple $[y_1, \cdots, y_n]$ along the name x and then behaves as P. On the other hand, $x([y_1, \cdots, y_n]).P$ atomically receives a tuple $[z_1, \cdots, z_n]$ along the name x and behaves as $P\{z_1/y_1, \cdots, z_n/y_n\}$. Thus $x([y_1, \cdots, y_n])$ works, in a sense, like the lambda prefix $\lambda y_1 \cdots y_n$. $P \mid Q$ represents concurrent agents P and Q interacting through names that they share. The statement $(\nu x)P$ means that x is private to P; this is interpreted as a fresh name declaration which is local to P. $\mathbf{0}$ designates an inactive agent; this term is no longer reduced. Simply α is often written as an abbreviation for $\alpha.\mathbf{0}$.

The following examples show how reduction proceeds (suppose that x does not occur freely in P):

$$
(\nu xy)(\overline{x}[y] \mid x([z]).P) \ \longrightarrow\ (\nu y)(P\{y/z\})
$$

$$(\nu xy)(\overline{x}[y] \mid y([x])) \longrightarrow \mathit{irreducible},$$

where $\longrightarrow$ means one-step reduction at the PPC level.

$\mathcal{A}$ is an agent identifier that is associated with the agent definition $\mathcal{A}[x] \stackrel{\text{def}}{=} P$. From the execution point of view, the agent identifier is regarded as a lazy macro expansion. The following is an example of an agent definition:

$$\begin{array}{l} f[p] \stackrel{\text{def}}{=} p([x]).(\overline{x}() \mid f[p]) \\ (\nu abp)(f[p] \mid \overline{p}[a] \mid a() \mid \overline{p}[b] \mid b()) \quad \rightarrow^\star \quad (\nu bp)(f[p] \mid \overline{p}[b] \mid b()) \\ \qquad\qquad\qquad\qquad\qquad\qquad\quad \rightarrow^\star \quad \mathbf{0} \end{array}$$

The notation $\stackrel{\text{def}}{=}$ is used for the definition of an agent including recursive call, whereas macro expansion (indicated as $=$) is used when an agent does not include recursive calls. A summation $\sum_{i \in I} \pi_i.P_i$ behaves like one of the P_i.

3 Translation

This section tells how to translate an FGHC$^-$ program to a PPC statements (agent definitions).

3.1 Term Representation in Polyadic π-Calculus

We start with the Milner's idea[4] for representing general data structures. Accordingly, the macro definitions for *cons*, *nil*, and *int* are given by

$$\begin{array}{rcl} \mathit{cons}[t,x,y] & = & t([c,n,i]).\overline{c}[x,y] \\ \mathit{nil}[t] & = & t([c,n,i]).\overline{n}() \\ \mathit{int}[t,v] & = & t([c,n,i]).\overline{i}[v] \end{array}$$

For instance, the *cons* definition represents the following protocol sequence: (1) a *cons* structure is located at t, (2) a reader agent sends a tuple $[c,n,i]$ as a probe via t, (3) after the tuple is received the name c is picked up, and (4) the tuple $[x,y]$ is returned to the reader via c. Since c is used for a channel, the reader agent can recognize that the sort is *cons* and the reply value is a two-element tuple. This method can encode a function symbol with any arity. Let a PPC statement in this form be called a *c-agent* and let $[c,n,i]$ be called a *sort tuple*. A sort tuple should include all the sorts occurring in an FGHC$^-$ program to be translated.

The following PPC statement reads out the value of a c-agent

$$\begin{array}{l} (\nu p)((\nu xy)(p([c,n,i]).\overline{c}[x,y]) \mid \underline{(\nu cni)(\overline{p}([c,n,i]) \mid c([a,b]).P)}) \\ \qquad\qquad\qquad\qquad\qquad\qquad \longrightarrow^\star \quad (\nu xy)(P\{x/a, y/b\}) \end{array}$$

Let the underlined fragment be called an *a-agent*.[1] Once an a-agent reads a c-agent, the c-agent terminates. For simplicity we introduce abbreviations for a c-agent and an a-agent:

$$\begin{array}{lrcl} \text{c-agent:} & p\langle \overline{s} \rangle[\vec{v}] & \equiv & p([\mathsf{st}]).\overline{s}[\vec{v}] \\ \text{a-agent:} & \overline{p}\langle s \rangle(\lambda \vec{v})Q & \equiv & \overline{p}[\mathsf{st}] \mid s([\vec{v}]).Q \end{array}$$

[1] *C* for c-agent and *a* for a-agent originates from concretion and abstraction [4].

Here p is the location of the agents, s is the sort and $\vec{v}$ the values, Q is a PPC statement, and st stands for a sort tuple. This notation eliminates the need to write sort tuples explicitly. Let us consider the reduction of a c-agent and an a-agent:

$$(\nu p)(\overline{p}\langle s\rangle(\lambda\vec{v})Q \mid p\langle\overline{s}\rangle[\vec{w}]) \longrightarrow^{\star} Q\{\vec{w}/\vec{v}\}$$

The scheme described in this paper interprets this reduction at the PPC level as follows: (1) the structure $s(\vec{w})$ is actively unified to a variable, (2) the variable is passively unified to $s(\vec{v})$, (3) the substitution $\{\vec{w}/\vec{v}\}$ is obtained, and (4) the execution of $Q\{\vec{w}/\vec{v}\}$ proceeds.

Lemma 3.1 *Given an a-agent $\overline{x}\langle f\rangle(\lambda\vec{y})P$, a corresponding c-agent of form $x'\langle\overline{f}\rangle[\vec{z}]$ s.t. $|\vec{z}| = |\vec{y}|$ can reduce the given a-agent. Similarly, given a c-agent $x\langle\overline{f}\rangle[\vec{z}]$, a corresponding a-agent of form $\overline{x'}\langle f\rangle(\lambda\vec{y})P$ s.t. $|\vec{y}| = |\vec{z}|$ can reduce the given c-agent.*

The following forwarder agent $\mathsf{fw}[r, w]$ represents a singly referenced variable:

$$\mathsf{fw}[r, w] \;=\; r(t).\overline{w}t$$

Any number of forwarders can cascade-connect a c-agent and an a-agent:

$$(\nu p_1 \cdots p_n)(\overline{p_1}\langle s\rangle(\lambda\vec{v})Q \mid \mathsf{fw}[p_1, p_2] \mid \cdots \mid \mathsf{fw}[p_{n-1}, p_n] \mid p_n\langle\overline{s}\rangle[\vec{w}]) \longrightarrow^{\star} Q\{\vec{w}/\vec{v}\}$$

To connect a c-agent and $n(\geq 2)$ a-agents, the scheme uses a new PPC agent $\mathsf{var}[r, w]$ defined as

$$\begin{aligned}
\mathsf{var}[r, w] = \;& (\nu cn)(\overline{w}[c, n, i] \mid (c([x, y]).\mathsf{rep_cons}[r, x, y] \\
& \qquad + n().\mathsf{rep_nil}[r] \\
& \qquad + i([v].\mathsf{rep_int}[r, v])) \\
\mathsf{rep_cons}[p, x, y] \stackrel{\text{def}}{=} \;& p([c, n, i]).(\overline{c}[x, y] \mid \mathsf{rep_cons}[p, x, y]) \\
\mathsf{rep_nil}[p] \stackrel{\text{def}}{=} \;& p([c, n, i]).(\overline{n}() \mid \mathsf{rep_nil}[p]) \\
\mathsf{rep_int}[p, v] \stackrel{\text{def}}{=} \;& p([c, n, i]).(\overline{i}[v] \mid \mathsf{rep_int}[p, v])
\end{aligned}$$

Here the sort tuple is $[c, n, i]$, and $\mathsf{var}[r, w]$ can be cascaded as $\mathsf{fw}[r, w]$. *in* and *out* occurrences of a variable respectively correspond to r and w for $\mathsf{var}[r, w]$, since the w channel works as an outlet of data and the r channel works as inlet (Fig. 1). The agent

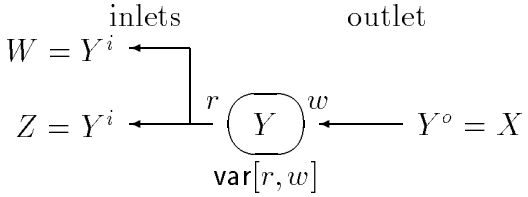

Figure 1: Moded logical variable and $\mathsf{var}[r, w]$

$\mathsf{var}[r, w]$ can synchronize write and read operations and duplicate the data written via the w channel as many times as read operations are issued. The agents whose symbol names begin with $\mathsf{rep_}$ are responsible for the data duplication.

Lemma 3.2 *Let each $\mathsf{au}[x, y]$ be either $\mathsf{fw}[x, y]$ or $\mathsf{var}[x, y]$. Given a c-agent $x\langle\overline{f}\rangle[\vec{z}]$, let Q be $(\nu x' x_1 \cdots x_m)$ $(\overline{x'}\langle f\rangle(\lambda\vec{y})P \mid \mathsf{au}[x', x_1] \mid \cdots \mid \mathsf{au}[x_m, x])$ $(m \geq 0,\ |\vec{y}| = |\vec{z}|)$. Then, to the c-agent, Q behaves the same as $\overline{x}\langle f\rangle(\lambda\vec{y})P$ does. That is, $(\nu x)(Q \mid x\langle\overline{f}\rangle[\vec{z}]) \rightarrow P\{\vec{z}/\vec{y}\}$. Inversely, given an a-agent $\overline{x}\langle f\rangle(\lambda\vec{z})P$, let R be $(\nu x' x_1 \cdots x_m)$ $(\mathsf{au}[x, x_1] \mid \cdots \mid \mathsf{au}[x_m, x'] \mid x'\langle\overline{f}\rangle[\vec{z}])$ $(m \geq 0,\ |\vec{z}| = |\vec{y}|)$. Then, to the a-agent, R behaves the same as $x\langle\overline{f}\rangle[\vec{z}]$ does. That is, (νx) $(R \mid \overline{x}\langle f\rangle(\lambda\vec{y})P) \rightarrow P\{\vec{z}/\vec{y}\}$.*

3.2 Translation Rules

The notation $[\![T]\!]p$ represents a PPC statement to which the FGHC$^-$ term T is translated and which includes the name p added by the translator. Note that $[\![\cdot]\!]$ depends on context.

T1: Clauses An entire program in FGHC$^-$ is translated to an agent definition, $clause[p]$, in PPC. That is, given a set of all clauses $c_1 \cdots c_m$ of all predicates,

$$clause[p] \stackrel{def}{=} (\nu cr)(\mathsf{var}[r,p] \mid \overline{c}() \mid [\![c_1]\!]rc \mid \cdots \mid [\![c_m]\!]rc)$$

is created. The definition clauses for different predicates are merged in an agent definition. Here $\mathsf{var}[r,p]$ duplicates a predicate invocation as many times as the number of definition clauses.

T2: Head For this translation the vocabulary with which source programs are written in FGHC$^-$ is used as it is as the sort names of PPC. The FGHC$^-$ variables, however, are described here in capital letters and the PPC names are described in small letters; e.g., the variables of FGHC$^-$, $\vec{A}$, correspond to the sort names of PPC, $\vec{a}$. Each clause c_i is translated to $[\![c_i]\!]rc$.

$$\begin{aligned}[\![u(\vec{A}^{\mu}) :\!- g \mid \vec{b}]\!]rc &= (\nu \mathsf{st})(\overline{r}[\mathsf{st}] \mid u([\vec{a}]).[\![g \mid \vec{b}]\!]c) \\ &\equiv \overline{r}\langle u\rangle(\lambda \vec{a})[\![g \mid \vec{b}]\!]c\end{aligned}$$

Here $\vec{b}$ stands for $b_1, \cdots, b_n$, and the translation of a head literal works as an a-agent. Note that a commit operator of FGHC$^-$ is distinguished from parallel composition of PPC, although the both are indicated by '|'.

T3: Guard The translation of $g \mid \vec{b}$ is given by

$$\begin{aligned}[\![true \mid \vec{b}]\!]c &= c().[\![\vec{b}]\!] \\ [\![A^i_j = f(\vec{D}^{\mu}) \mid \vec{b}]\!]c &= (\nu \mathsf{st})(\overline{a_j}[\mathsf{st}] \mid f([\vec{d}]).c().[\![\vec{b}]\!]) \\ &\equiv \overline{a_j}\langle f\rangle(\lambda \vec{d})(c().[\![\vec{b}]\!]) \\ [\![A^i_j = A^i_k \mid \vec{b}]\!]c &= (\nu s)(\mathsf{p_unif}[s, a_j, a_k] \mid s().c().[\![\vec{b}]\!]) \\ [\![g_builtin(\vec{E}^i) \mid \vec{b}]\!]c &= (\nu s)(\mathsf{g_builtin}[s, \vec{e}] \mid s().c().[\![\vec{b}]\!])\end{aligned}$$

The term decomposition by the passive unification, $A^i_j = f(\vec{D}^{\mu})$, is implemented by an a-agent; then the obtained substitution is applied to $[\![\vec{b}]\!]$. The passive unification of two variables is translated to a system-supported agent $\mathsf{p_unif}[s, a, b]$ (details in Section 3.3 and Definition 4.6). If a and b are instantiated to the same ground structure, $\mathsf{p_unif}[s, a, b]$ sends a signal via s. Thus the passive unification of our scheme conforms to that of KL1 [9].

The prefix $c()$ implements a commit operator. Since all the clauses of all predicates are merged into an agent $clause[p]$, in general, more than one $c()$ waits for a signal from $\overline{c}()$ introduced in T1.

T4: Body Goals The translation of a goal and body part, $b_1, \cdots, b_n$, is given by

$$\begin{aligned}[\![\vec{b}]\!] = (\nu \vec{w})((\nu p_1)([\![b_1]\!]p_1 \mid clause[p_1]) \mid \cdots \mid (\nu p_m)([\![b_m]\!]p_m \mid clause[p_m]) \\ \mid [\![b_{m+1}]\!] \mid \cdots \mid [\![b_n]\!])\end{aligned}$$

Let $\vec{W}$ be a set of c-vars moded *out*, except for the variables occurring either in the head or in the guard. The $\vec{w}$ above corresponds to the $\vec{W}$. Body goals $b_1 \cdots b_m$ $(0 \leq m \leq n)$ are user-defined predicates.

T5: Predicate and Active Unification When a body goal b_i does not include c-vars, the translation rule is given by

$$\begin{array}{rcl} [\![true]\!] & = & \mathbf{0} \\ [\![u(\vec{R}^{\mu})]\!]p & = & p([\mathsf{st}]).\overline{u}[\vec{r}] \ \equiv \ p\langle\overline{u}\rangle[\vec{r}] \\ [\![b_builtin(\vec{R}^{\mu})]\!] & = & \mathsf{b_builtin}[\vec{r}] \\ [\![Y_i^o = f(\vec{R}^{\mu})]\!] & = & y_i([\mathsf{st}]).\overline{f}[\vec{r}] \ \equiv \ y_i\langle\overline{f}\rangle[\vec{r}] \\ [\![Y_j^o = X_k^i]\!] & = & \mathsf{fw}[y_j, x_k] \end{array}$$

A predicate is obviously implemented by a c-agent. The c-agent $[\![b_i]\!]p_i$ $(0 \leq i \leq m)$ and the a-agent $clause[p]$ defined in T1 form a predicate invocation.

T6: Communication Variable When c-vars occur at the arguments' positions in T5, $\mathsf{var}[r, w]$ should be inserted properly. Here we use $t\boxed{x}$ to designate the term t including each free occurrence of the variable x. If Y is a c-var and $[\![T\boxed{Y^o}]\!] = t\boxed{y}$ in T5, then

$$[\![T\boxed{Y^c}]\!] = (\nu v)(\mathsf{var}[y, v] \mid t\boxed{v})$$

Two examples are

$$\begin{array}{rcl} [\![Y_i^c = f(\vec{R}^{\mu})]\!] & = & (\nu v)(\mathsf{var}[y_i, v] \mid v\langle\overline{f}\rangle[\vec{r}]) \\ [\![Y_j{}^c = X_k^i]\!] & = & (\nu v)(\mathsf{var}[y_j, v] \mid \mathsf{fw}[v, x_k]) \ \ (\rightarrow^{\star} \mathsf{var}[y_j, x_k]) \end{array}$$

The arguments of predicates and functions are divided into c-vars moded *out* ($\vec{Z}^c$) and the rest ($\vec{R}^{\mu}$):

$$[\![u(\vec{R}^{\mu}, \vec{Z}^c)]\!]p \ = \ (\nu v_1 \cdots v_l)(p\langle\overline{u}\rangle[\vec{r}, v_1, \cdots, v_l] \mid \mathsf{var}[z_1, v_1] \mid \cdots \mid \mathsf{var}[z_l, v_l])$$

Since the occurrence of a c-var moded *out* acts as a source for one-to-n communication, $\mathsf{var}[r, w]$ duplicates on demand the data that the source supplies just once.

3.3 System-Supported PPC Agents

The translator must provide a runtime library that reflects the information of all sort names occurring in a given FGHC$^-$ program. The runtime library consists of $\mathsf{p_unif}[s, a, b]$ for passive unification of two variables, the guard built-in and body built-in predicates, $\mathsf{var}[r, w]$ and $\mathsf{fw}[r, w]$.

When a sort tuple is $[c, n, i]$ (implying $cons/2$, $nil/0$, $int/1$, respectively), $\mathsf{p_unif}[s, a, b]$ is defined as[2]

$$\begin{array}{l} \mathsf{p_unif}[s, a, b] \stackrel{\text{def}}{=} \ (\nu c_a n_a i_a c_b n_b i_b)(\overline{a}[c_a, n_a, i_a] \mid \overline{b}[c_b, n_b, i_b] \mid \\ \qquad ((c_a([x_a, y_a]).c_b([x_b, y_b]).(\nu s_x s_y)(\mathsf{p_unif}[s_x, x_a, x_b] \\ \qquad\qquad \mid \mathsf{p_unif}[s_y, y_a, y_b] \\ \qquad\qquad \mid s_x().s_y().\overline{s}())) \\ \qquad + (n_a().n_b().\overline{s}()) \\ \qquad + (i_a([v_a]).i_b([v_b]).\mathsf{p_unif}[s, v_a, v_b]))) \end{array}$$

[2]The actual $\mathsf{p_unif}[s, a, b]$ includes the code for treating a special case in which a and b refer to integers.

The guard built-in predicates, such as $X \leq 0$, $X > Y$ and $print(X)$, are translated to the agents of the form $\mathsf{g_builtin}[s, \vec{e}]$; $\vec{e}$ represents input arguments and s is a channel via which the guard built-in agent sends a signal for successful termination. The body built-in predicates, such as $modulo(X, Y, M)$ and $minus1(X, R)$, are of the form $\mathsf{b_builtin}[\vec{r}]$. Although this agent does not have a signal channel, $\vec{r}$ possibly includes input and output arguments. The definitions of $\mathsf{var}[r, w]$ and $\mathsf{fw}[r, w]$ are already given in Section 3.1.

Example (APPEND Program) The APPEND program in FGHC$^-$ is shown below:

$$\begin{array}{lll} app(X^i, Y^i, Z^o) & :- & X^i = nil \quad | \quad Y^i = Z^o. \\ app(X^i, Y^i, Z^o) & :- & X^i = cons(E^m, X_s^o) \quad | \quad Z^o = cons(E^{\overline{m}}, Z_s^i), app(X_s^i, Y^i, Z_s^o). \end{array}$$

Z_s in the second clause is a c-var, and its second occurrence is ${Z_s}^c$. The translation result is presented, using the notations for an a-agent and a c-agent below:

$$\begin{array}{rl} clause[v_0] \stackrel{\text{def}}{=} & (\nu v_1 v_2)(var[v_2, v_0] \mid \overline{v_1}() \\ & \quad \mid \overline{v_2}\langle app\rangle(\lambda xyz)(\overline{x}\langle nil\rangle(\lambda)(v_1().\mathsf{fw}[z, y])) \\ & \quad \mid \overline{v_2}\langle app\rangle(\lambda xyz)(\overline{x}\langle cons\rangle(\lambda e x_s)(v_1(). \\ & \qquad (\nu z_s)((\nu v_{17})((\nu v_{18})(v_{17}\langle\overline{app}\rangle[x_s, y, v_{18}] \mid \mathsf{var}[z_s, v_{18}]) \\ & \qquad\quad \mid clause[v_{17}]) \\ & \qquad \mid z\langle\overline{cons}\rangle[e, z_s])))) \end{array}$$

The names v_n are created during the translation; v_1, for example, implements a commit operator. Although the agent identifiers $\mathsf{var}[r, w]$ and $\mathsf{fw}[r, w]$ should be macro-expanded in place, for readability the result is shown above without macro expansion. The two clauses of FGHC$^-$ are merged into one agent definition. The active unification in the first clause $Y^i = Z^o$ is translated to $\mathsf{fw}[z, y]$, and the c-var Z_s in the second clause is translated to $\mathsf{var}[z_s, v_{18}]$.

4 π-Calculus Semantics

4.1 Ueda's Operational Semantics of GHC

Ueda's operational semantics of GHC is based on a transition system [8]. The configuration represents the computation states of an entire system and is defined as $\langle \mathcal{B}, \mathcal{C}\rangle : \mathcal{V}$. $\mathcal{B}$ is a multiset of all goals, such as $p(X)$, $Y = w(Z)$. $\mathcal{C}$ is the constraints on the variables generated by active unification. $\mathcal{V}$ is $var(\mathcal{B} \cup \mathcal{C})$. $\mathcal{P}$ is a program which is a set of clauses. Using the notation of a transition system, we can express the operational meaning of goal reduction as follows:

$$\frac{\langle B_1, C_1\rangle : V_1 \rightarrow \langle B_1', C_1'\rangle : V_1'}{\langle B_1 \cup B_2, C_1\rangle : V_1 \rightarrow \langle B_1' \cup B_2, C_1'\rangle : V_1'} \qquad \textbf{U1}$$

$$\frac{\mathcal{P} \vdash \langle \{\overline{b} = \overline{h_i}\} \cup G_i, C\rangle : (V \cup \mathcal{V}_{(h_i, G_i)}) \stackrel{\star}{\rightarrow} \langle \emptyset, C \cup C_g\rangle : V'}{\{h_i :- \ G_i \mid B_i\} \cup \mathcal{P} \vdash \langle \{b\}, C\rangle : V \rightarrow \langle B_i, C \cup C_g\rangle : (V' \cup \mathcal{V}_{B_i})} \qquad \textbf{U2}$$

$$\text{if } \mathcal{E} \models \forall(C \Rightarrow \delta\mathcal{V}_b.C_g) \text{ and } \mathcal{V}_{(h_i, G_i, B_i)} \cap V = \emptyset,$$
$$\delta\mathcal{V}_b \equiv \exists(var(C_g) \setminus var(b))$$

$$\frac{}{\langle s = t, \ C\rangle : V \rightarrow \langle \emptyset, C \cup \{s = t\}\rangle : V} \qquad \textbf{U3}$$

Rule U1 shows that goals are reduced concurrently, and rule U2 shows the reduction by one step, including both passive unification and the creation of new goals. Here $\mathcal{E}$ means Clark's equality theory. Rule U3 shows active unification.

In FGHC$^-$, $\mathcal{B}$ for $\langle \mathcal{B}, \mathcal{C} \rangle : \mathcal{V}$ consists of $u(\vec{X})$, $X^o = Y^i$, and $X^o = f(\vec{Y}^\mu)$. $\mathcal{C}$ consists of $X^o = Y^i$, $X^o = f(\vec{Y}^\mu)$, $X^i = f(\vec{Y}^\mu)$, and $X^i = Y^i$. However, $\langle \{\overline{b} = \overline{h_i}\} \cup G_i, C \rangle$ of U2 is slightly special; $\mathcal{B}$ additionally includes $X^i = f(\vec{Y}^\mu)$ and $X^i = Y^i$. $\mathcal{C}$ is always well-moded in a sense that each variable has at most one writer.

4.2 Moded Ground Equality Theory

The moded ground equality theory (*MGE*) defines the unification carried out in FGHC$^-$. A moded variables V^μ is distiguished from $V^{\overline{\mu}}$ (inversely moded). Moded terms s and t are of form, Z^μ or $f(Z_1^{\mu_1}, \cdots, Z_n^{\mu_n})$ $(n \geq 0)$.

Definition 4.1 MGE *is defined as* $(n \geq 0)$*:*

$$\forall(X^o = f(U_1^{\mu_1}, \cdots, U_n^{\mu_n}) \wedge X^i = f(V_1^{\overline{\mu}_1}, \cdots, V_n^{\overline{\mu}_n}) \Rightarrow U_1^{\mu_1} = V_1^{\overline{\mu}_1} \wedge \cdots \wedge U_n^{\mu_n} = V_n^{\overline{\mu}_n}) \quad \textbf{M1}$$

$$\forall(U_1^i = V_1^i \wedge \cdots \wedge U_n^i = V_n^i \wedge X^o = f(U_1^{\mu_1}, \cdots, U_n^{\mu_n}) \wedge Y^o = f(V_1^{\mu'_1}, \cdots, V_n^{\mu'_n}) \Rightarrow X^i = Y^i) \quad \textbf{M2}$$

$$\forall(X^i = Y^i \Rightarrow U_1^i = V_1^i \wedge \cdots \wedge U_n^i = V_n^i \wedge X^o = f(U_1^{\mu_1}, \cdots, U_n^{\mu_n}) \wedge Y^o = f(V_1^{\mu'_1}, \cdots, V_n^{\mu'_n}) \quad \textbf{M3}$$

$$\forall(Y^\mu = X^{\overline{\mu}} \wedge X^\mu = t \Rightarrow Y^\mu = t) \textit{ for each } t \textit{ } (Z^{\overline{\mu}} \textit{ or } f(Z_1^{\mu_1}, \cdots, Z_n^{\mu_n})) \quad \textbf{M4}$$

$$\forall(s = t \Rightarrow t = s) \textit{ for each } s \textit{ and } t \quad \textbf{M5}$$

$$\forall(\neg(X^\mu = t)) \textit{ for each } t \textit{ containing } X^{\overline{\mu}} \quad \textbf{M6}$$

One can obtain the logical meaning of *MGE* by removing the mode indications. Note that $X^o = X^i$ is ill-moded. M2 and M3 guarantee the groundness of the unification $X^i = Y^i$. The logical axioms used here should also be moded. For instance, from the substitution axiom we can easily derive

$$\forall(X^o = f(U_1^{\mu_1}, \cdots, U_n^{\mu_n}) \Rightarrow \exists V \; X^i = f(V_1^{\overline{\mu}_1}, \cdots, V_n^{\overline{\mu}_n})) \quad \textbf{A1}$$

Lemma 4.2 *Suppose that* t *is a moded formula in which* $\vec{X}$ *freely occur,* $t \wedge \{\vec{Y}^\mu = \vec{X}^{\overline{\mu}}\}$ *has a well-moding, and* $var(t) \cap var(\vec{Y}) = \emptyset$*. Then,* $\forall(t\{\vec{Y}/\vec{X}\} \Leftrightarrow \exists \vec{X} \; t \wedge \{\vec{Y}^\mu = \vec{X}^{\overline{\mu}}\})$.

Lemma 4.3 *Suppose that* t *is a moded term,* C *is a set of moded constraints,* $var(t) \cap var(C) = \emptyset$*,* $\vec{X} \in var(t)$*,* $\vec{Y} \in var(C)$*, and* $C \wedge (\vec{Y}^\mu = \vec{X}^{\overline{\mu}})$ *has a well-moding. Then,* $\forall(C \Rightarrow t\{\vec{Y}/\vec{X}\}) \Leftrightarrow \forall(C \wedge (\vec{Y}^\mu = \vec{X}^{\overline{\mu}}) \Rightarrow t)$.

Proof $\forall(C \Rightarrow t\{\vec{Y}/\vec{X}\})$ iff $\forall(C \Rightarrow t\{\vec{Y}/\vec{X}\})\{\vec{X}/\vec{Y}\}$ iff $\forall(C\{\vec{X}/\vec{Y}\} \Rightarrow t)$ iff $\forall(\exists \vec{Y} C \wedge (\vec{Y}^\mu = \vec{X}^{\overline{\mu}})$(immediately from Lemma 4.2) $\Rightarrow t)$ iff $\forall(C \wedge (\vec{Y}^\mu = \vec{X}^{\overline{\mu}}) \Rightarrow t)$ $\square$

4.3 Configuration in π-Calculus Semantics

Reminder The translation from FGHC$^-$ terms to PPC statements is denoted by $[\![\cdot]\!]$ and defined as follows:

$$
\begin{array}{lcl}
[\![u(\vec{X}^{\mu})]\!] & = & (\nu p)(p\langle\overline{u}\rangle[\vec{x}] \mid clause[p]) \\
[\![X^{o} = f(\vec{U}^{\mu})]\!] & = & x\langle\overline{f}\rangle[\vec{u}] \\
 & & x \text{ does not occur as the argument of an agent } clause[p] \\
[\![X^{o} = Y^{i}]\!] & = & \mathsf{var}[x,y] \text{ if the number of occurrences of } X^{i} \text{ is } > 1 \\
 & & \mathsf{fw}[x,y] \text{ if the number of occurrences of } X^{i} \text{ is } 1 \\
[\![X^{i} = f(\vec{V}^{\mu}) \mid Q]\!] & = & \overline{x}\langle f\rangle(\lambda\vec{v})[\![Q]\!] \\
[\![X^{i} = Y^{i} \mid Q]\!] & = & (\nu s)(\mathsf{p_unif}[s,x,y] \mid s().[\![Q]\!])
\end{array}
$$

These $[\![\cdot]\!]$'s are independent from the context in which they are used. We omit the receiver $c()$ for a commit operator for simplicity.

Definition 4.4 *Corresponding to the configuration of Ueda's semantics* $\langle\mathcal{B},\mathcal{C}\rangle : \mathcal{V}$, the configuration in the π-calculus semantics *is defined as* $\langle[\![\mathcal{B}]\!],[\![\mathcal{C}]\!]\rangle : \mathcal{N}$, *where* $[\![\mathcal{B}]\!]$ *is a set of translated moded goals and* $[\![C]\!]$ *is a set of translated moded constraints. Here* $\mathcal{N}$ *is* $name([\![\mathcal{B}]\!] \cup [\![\mathcal{C}]\!])$. $name(S)$ *stands for names in* S.

Definition 4.5 *The translation from FGHC terms to PPC statements is denoted by* $[\![\cdot]\!]$, *and the inverse translation by* $[\![\cdot]\!]^{-1}$. *Given a configuration in the* π*-calculus semantics* $\langle\mathcal{A},\mathcal{I}\rangle : \mathcal{N}$, $\langle\mathcal{A},\mathcal{I}\rangle : \mathcal{N}$ *is* well-formed *if* $[\![[\![\mathcal{A}]\!]^{-1}]\!] = \mathcal{A}$ *and* $[\![[\![\mathcal{I}]\!]^{-1}]\!] = \mathcal{I}$. $\mathcal{A}$ *and* $\mathcal{I}$ *are also well-formed.*

Note that, given a configuration in FGHC^{-} , $[\![[\![\mathcal{B}]\!]]^{-1} = \mathcal{B}$ and $[\![[\![\mathcal{C}]\!]]^{-1} = \mathcal{C}$ always hold. It is obvious that $[\![\cdot]\!]^{-1}$ is also a function because of the uniqueness of the translation rules listed in the above Reminder.

Definition 4.6 $\mathsf{p_unif}[s,a,b]$ *recursively checks whether* a *and* b *are bound to the same structure and implements M2. The PPC code for the recursive decomposition of an* n*-ary function* f $(n \geq 0)$ *is given by*

$$
\begin{array}{lcl}
\mathsf{p_unif}[s,a,b] & \stackrel{def}{=} & \overline{a}\langle f\rangle(\lambda\vec{u})(\overline{b}\langle f\rangle(\lambda\vec{v}) \\
 & & \quad (\nu s_1 \cdots s_n)(\mathsf{p_unif}[s_1,u_1,v_1] \mid \cdots \mid \mathsf{p_unif}[s_n,u_n,v_n] \\
 & & \quad\quad \mid s_1().\cdots.s_n().\overline{s}()))
\end{array}
$$

The actual code of $\mathsf{p_unif}[s,a,b]$ presented in Section 3.3 is constructed based on Definition 4.6.

Lemma 4.7 *For a set of moded constraints* C, *let* $\mathcal{I}$ *be* $[\![C]\!]$ *and well-formed.*

(1) $MGE \models \forall(C \Rightarrow \exists\vec{Y}\ X^{i} = f(\vec{Y}^{\mu}))$ *iff* $\mathcal{I} \mid \overline{x}\langle f\rangle(\lambda\vec{y})Q \rightarrow^{\star} \mathcal{I}' \mid Q\{\vec{y'}/\vec{y}\}$

(2) $MGE \models \forall(C \Rightarrow X^{i} = Y^{i})$ *iff* $\mathcal{I} \mid (\nu s)(\mathsf{p_unif}[s,x,y] \mid s()) \rightarrow^{\star} \mathcal{I}'$

Here $\mathcal{I}' \subseteq \mathcal{I}$.

$\mathsf{p_unif}[s,x,y]$ *is an agent identifier introduced by Definition 4.6.*

Proof (1) Suppose $\mathcal{I} \mid \overline{x}\langle f\rangle(\lambda\vec{y})Q \rightarrow^{\star} \mathcal{I}' \mid Q\{\vec{y'}/\vec{y}\}$. To reduce the a-agent $\overline{x}\langle f\rangle(\lambda\vec{y})Q$, there exists a corresponding c-agent in $\mathcal{I}$ by Lemma 3.1. Thus, by Lemma 3.2, for some $\vec{y'}$, $\mathsf{au}[x,x'] \mid \cdots \mid x''\langle\overline{f}\rangle[\vec{y'}] \subseteq \mathcal{I}$. Note that each $\mathsf{au}[r,w]$ stands for $\mathsf{fw}[r,w]$ or $\mathsf{var}[r,w]$. Since $\mathcal{I}$ is well-formed, by $[\![\cdot]\!]^{-1}$, we have $X^{o} = X'^{i} \wedge \cdots \wedge X''^{o} = f(\vec{Y'}^{\mu}) \subseteq C$. Hence, $MGE \models \forall(C \Rightarrow X^{o} = f(\vec{Y'}^{\mu}))$. Therefore, A1 leads to $MGE \models \forall(C \Rightarrow \exists\vec{Y}.X^{i} = f(\vec{Y}^{\mu}))$ The proof of the only-if part of (1) is as follows. Examining the forms of unification

included in $\mathcal{C}$ (Section 4.1), $MGE \models \forall(C \Rightarrow X^o = f(\vec{Y'}^{\overline{\mu}}))$ should hold. Hence, $X^o = X'^o \wedge \cdots X''^o = f(\vec{Y'}^{\overline{\mu}}) \subseteq C$. By $[\![\cdot]\!]$, $\mathsf{fw}[x,x']$ or $\mathsf{var}[x,x'] \mid \cdots \mid x''\langle\overline{f}\rangle[\vec{y'}] \subseteq \mathcal{I}$. Thus, by the PPC level reduction, $\mathcal{I} \mid \overline{x}\langle f\rangle(\lambda\vec{y})Q \rightarrow \mathcal{I}' \mid Q\{\vec{y'}/\vec{y}\}$.

The proof of (2) is by induction on the depth of a ground structure. Consider the base case in which the depth is 1. Assume $\mathsf{p_unif}[s,x,y] \stackrel{def}{=} \overline{x}\langle f\rangle(\lambda)(\overline{y}\langle f\rangle(\lambda)\overline{s}())$ and $X^o = f \wedge Y^o = f \Leftrightarrow X^i = Y^i$ (M2 and M3). Suppose that $\mathcal{I} \mid (\nu s)(\mathsf{p_unif}[s,x,y] \mid s()) \rightarrow^\star \mathcal{I}'$. It follows from Lemmas 3.1, 3.2, and the definition of $\mathsf{p_unif}[s,x,y]$ that $\mathsf{au}[x,x'] \mid \cdots \mid x''\langle\overline{f}\rangle[] \mid \mathsf{au}[y,y'] \mid \cdots \mid y''\langle\overline{f}\rangle[] \subseteq \mathcal{I}$. Using $[\![\cdot]\!]^{-1}$, we have $X^o = X'^i \wedge \cdots \wedge X''^o = f \wedge Y^o = Y'^i \wedge \cdots \wedge Y''^o = f \subseteq C$. Thus, it follows from M2 that $MGE \models \forall(C \Rightarrow X^i = Y^i)$. Conversely suppose $MGE \models \forall(C \Rightarrow X^i = Y^i)$. Using M3, we should have $MGE \models \forall(C \Rightarrow X^o = f \wedge Y^o = f)$. The rest of the proof is similar to that of the if part.

Now, consider the inductive case in which the depth is n. Assume that $\mathcal{I} \mid (\nu s_j)(\mathsf{p_unif}[s_j,u_j,v_j] \mid s_j()) \rightarrow^\star \mathcal{I}'_j$ *iff* $MGE \models \forall(C \Rightarrow U^i_j = V^i_j)$ for each j. Suppose $\mathcal{I} \mid (\nu s)(\mathsf{p_unif}[s,x,y] \mid s()) \rightarrow^\star \mathcal{I}'$. From the definition of $\mathsf{p_unif}[s,x,y]$, $\mathsf{au}[x,x'] \mid \cdots \mid x''\langle\overline{f}\rangle[\vec{u}] \mid \mathsf{au}[y,y'] \mid \cdots \mid y''\langle\overline{f}\rangle[\vec{v}] \subseteq \mathcal{I}$ should hold by Lemmas 3.1 and 3.2. Let R be $\mathcal{I}'' \mid (\nu s s_1 \cdots s_n)\ (\mathsf{p_unif}[s_1,u_1,v_1] \mid \cdots \mid \mathsf{p_unif}[s_n,u_n,v_n] \mid s_1().\cdots.s_n().\overline{s}() \mid s())$. Then $\mathcal{I} \mid (\nu s)\ (\mathsf{p_unif}[s,x,y] \mid s()) \rightarrow^\star R$. Hence, by the induction hypothesis and $[\![\cdot]\!]^{-1}$, that R is reduced to $\mathcal{I}''' \mid \mathcal{I}'_1 \mid \cdots \mid \mathcal{I}'_n\ (= \mathcal{I}')$ and that $MGE \models \forall(C \Rightarrow U^i_1 = V^i_1 \wedge \cdots \wedge U^i_n = V^i_n \wedge X^o = f(\vec{U}^\mu) \wedge Y^o = f(\vec{V}^\mu))$ are equivalent. Therefore it follows from M2 that $MGE \models \forall(C \Rightarrow X^i = Y^i)$. The proof of the only-if part of (2) is as follows. From M3, $\models \forall(C \Rightarrow U^i_1 = V^i_1 \wedge \cdots \wedge U^i_n = V^i_n \wedge X^o = f(\vec{U}^\mu) \wedge Y^o = f(\vec{V}^\mu))$. The fragment, $U^i_1 = V^i_1 \wedge \cdots \wedge U^i_n = V^i_n$, is equivalent to $\mathcal{I} \mid (\nu s_j)(\mathsf{p_unif}[s_j,u_j,v_j] \mid s_j()) \rightarrow^\star \mathcal{I}'_j$ (for all j) from the hypothesis of induction. The fragment, $X^o = f(\vec{U}^\mu) \wedge Y^o = f(\vec{V}^\mu)$, is translated to $x\langle\overline{f}\rangle[\vec{u}] \mid y\langle\overline{f}\rangle[\vec{v}] \subseteq \mathcal{I} = [\![C]\!]$ by T5. Using Definition 4.6, $\mathcal{I} \mid (\nu s)(\mathsf{p_unif}[s,x,y] \mid s()) \rightarrow^\star (\nu s_1 \cdots s_n)(\mathsf{p_unif}[s_1,u_1,v_1] \mid \cdots \mid \mathsf{p_unif}[s_n,u_n,v_n]) \rightarrow^\star \mathcal{I}'$

Note that the groundness of a structure prevents infinite induction. □

The next three subsections show that the PPC level reduction of the translated PPC statements obeys the transition rules of an FGHC$^-$ configuration: U1, U2, and U3. When not causing confusion, we write configurations without variables and names, such as $\langle\mathcal{B},\mathcal{C}\rangle$ and $\langle\mathcal{A},\mathcal{I}\rangle$.

4.4 Concurrent Rewriting

U1$^-$(the FGHC$^-$ version of U1) is the same as U1. By using $[\![\cdot]\!]$ of Definition 4.4, we can give the PPC version of U1 as

$$\frac{\langle [\![B_1]\!],\mathcal{I}\rangle : N \rightsquigarrow \langle [\![B'_1]\!],\mathcal{I}'\rangle : N'}{\langle [\![B_1 \cup B_2]\!],\mathcal{I}\rangle : N \rightsquigarrow \langle [\![B'_1 \cup B_2]\!],\mathcal{I}'\rangle : N'} \quad \mathbf{U1}^\pi$$

Here $\mathcal{I} = [\![C_1]\!], \mathcal{I}' = [\![C'_1]\!]$, and $\mathcal{I}$ and $\mathcal{I}'$ are well-formed. Both the antecedents of U1$^-$and U1$^\pi$represent the reduction of a body goal B_1 by one step. However, the reduction by one step in FGHC$^-$ generally corresponds to the reduction by more than one step in PPC. Thus the transition from a well-formed configuration to another well-formed configuration at the PPC level is indicated by $\rightsquigarrow$.

Theorem 4.8 *U1$^-$ iff U1$^\pi$.*

Proof Given an FGHC$^-$ program, T4 and Definition 4.4 translate $\langle B_1 \cup B_2, C\rangle$ to $\langle(\nu\vec{w})([\![B_1]\!] \mid [\![B_2]\!]),\mathcal{I}\rangle$. This configuration is obviously well-formed. From the antecedent of U1$^\pi$, $\langle(\nu\vec{w})([\![B_1]\!] \mid [\![B_2]\!]),\mathcal{I}\rangle$ is reduced to $\langle(\nu\vec{w})([\![B'_1]\!] \mid [\![B_2]\!]),\mathcal{I}'\rangle$ at the

PPC level. Here, $[\![B_1']\!]$ corresponds to the result of the reduction by one step at the FGHC$^-$ level. Since $(\nu\vec{w})([\![B_1']\!] \mid [\![B_2]\!])$ is well-formed, $\langle(\nu\vec{w})([\![B_1']\!] \mid [\![B_2]\!]), \mathcal{I}'\rangle$ is equivalent to $\langle [\![B_1' \cup B_2]\!], \mathcal{I}'\rangle$ and, further, is equivalent to $\langle B_1' \cup B_2, C_1'\rangle : V_1'$ by $[\![\cdot]\!]^{-1}$ and $[\![\cdot]\!]$. □

4.5 One-Step Reduction

U2 is specialized for FGHC$^-$ as follows:

$$\frac{\langle \{\overline{u(\vec{R}^{\mu})} = \overline{h(\vec{A}^{\overline{\mu}})}\} \cup g, C\rangle : (V \cup \mathcal{V}_{(h,g)}) \xrightarrow{\star} \langle \emptyset, C \cup C_g\rangle : V'}{\{h(\vec{A}^{\overline{\mu}}) \coloneq g \mid \vec{b}\} \vdash \langle \{u(\vec{R}^{\mu})\}, C\rangle : V \rightarrow \langle \vec{b}, C \cup C_g\rangle : (V' \cup \mathcal{V}_{\vec{b}})} \quad \mathbf{U2^-}$$

$$\text{if } MGE \models \forall.(C \Rightarrow \delta\mathcal{V}_u.C_g) \text{ and } \mathcal{V}_{(h,g,\vec{b})} \cap V = \emptyset$$
$$g \text{ is either } A_j^i = f(\vec{D}^{\mu}) \text{ or } A_j^i = A_k^i$$

The PPC version of U2$^-$is

$$\frac{\langle [\![P^o = u(\vec{R}^{\mu}) \cup P^i = h(\vec{A}^{\overline{\mu}}) \cup g]\!], \mathcal{I}\rangle : (N \cup \mathcal{N}_{(h,g)}) \rightsquigarrow \langle \mathbf{0}\theta, \mathcal{I}'\rangle : N'}{\{h(\vec{A}^{\overline{\mu}}) \coloneq g \mid \vec{b}\} \vdash \langle [\![u(\vec{R}^{\mu})]\!], \mathcal{I}\rangle : N \rightsquigarrow \langle [\![\vec{b}]\!]\theta, \mathcal{I}'\rangle : (N' \cup \mathcal{N}_{\vec{b}})} \quad \mathbf{U2^{\pi}}$$
$$\text{if } \mathcal{I} \mid [\![g]\!]\{\vec{r}/\vec{a}\} \rightarrow^{\star} \mathcal{I}', \text{ where } \mathcal{I}' \subseteq \mathcal{I} \text{ and } \mathcal{N}_{(h,g,\vec{b})} \cap N = \emptyset$$

$$g \text{ is either } A_j^i = f(\vec{D}^{\mu}) \text{ or } A_j^i = A_k^i. \quad \theta = \begin{cases} \{\vec{r}/\vec{a}, \vec{d'}/\vec{d}\} & \text{if } g \text{ is } A_j^i = f(\vec{D}^{\mu}) \\ \{\vec{r}/\vec{a}\} & \text{if } g \text{ is } A_j^i = A_k^i \end{cases}$$

Since '$\overline{\cdot}$' denotes a one-to-one mapping from the set of predicates to the set of fuctions, $\overline{u(\vec{R}^{\mu})} = \overline{h(\vec{A}^{\overline{\mu}})}$ means just active and passive unification, $P^o = u(\vec{R}^{\mu}) \wedge P^i = h(\vec{A}^{\overline{\mu}})$. As a result, by M1, $\vec{R}^{\mu} = \vec{A}^{\overline{\mu}}$ is put into C_g.

The transition '$\xrightarrow{\star}$' in the antecedent of U2$^-$is achieved by U3$^-$. That is, any active unification can be put into the set of constraints. Therefore, in U2$^-$we eventually obtain the configuration $\langle \emptyset, C \cup \{\vec{R}^{\mu} = \vec{A}^{\overline{\mu}}, A_j^i = f(\vec{D}^{\mu})\}\rangle$ or $\langle \emptyset, C \cup \{\vec{R}^{\mu} = \vec{A}^{\overline{\mu}}, A_j^i = A_k^i\}\rangle$.

Correspondingly, in U2$^{\pi}$by Lemma 4.3, M5, and $[\![\cdot]\!]$, we should obtain $\langle \mathbf{0}\{\vec{r}/\vec{a}, \vec{d'}/\vec{d}\}, \mathcal{I}\rangle$ or $\langle \mathbf{0}\{\vec{r}/\vec{a}\}, \mathcal{I}\rangle$. However, at the PPC level we can only reduce $\mathcal{I} \mid [\![P^o = u(\vec{R}^{\mu}) \wedge P^i = h(\vec{A}^{\overline{\mu}}) \wedge g]\!]$ to $\mathcal{I} \mid [\![g]\!]\{\vec{r}/\vec{a}\}$. It is not reduced further, because there is no information to proceed with the reduction of g at the PPC level. Thus the following proposition is needed.

Proposition 4.9 *Let $\mathcal{I}$ be $[\![C]\!]$ and well-formed. $\mathcal{I} \mid [\![g]\!]\{\vec{r}/\vec{a}\} \rightarrow^{\star} \mathcal{I}'$ iff $MGE \models \forall.(C \Rightarrow \delta\mathcal{V}_u.C_g)$.*

Here $\delta\mathcal{V}_u C_g = \begin{cases} \exists\vec{A}\exists\vec{D}\ \vec{R}^{\mu} = \vec{A}^{\overline{\mu}} \wedge A_j^i = f(\vec{D}^{\mu}) & \text{if } g \text{ is } A_j^i = f(\vec{D}^{\mu}) \\ \exists\vec{A}\ \vec{R}^{\mu} = \vec{A}^{\overline{\mu}} \wedge A_j^i = A_k^i & \text{if } g \text{ is } A_j^i = A_k^i \end{cases}$

$var(\vec{R}) \cup var(\vec{A}) \cup var(\vec{D})$ are distinct.

Proof Lemma 4.2 equivalently rewrites $MGE \models \forall.(C \Rightarrow \delta\mathcal{V}_u.C_g)$ to $MGE \models \forall.(C \Rightarrow \exists\vec{D}.R_j^i = f(\vec{D}^{\mu}))$ if g is $A_j^i = f(\vec{D}^{\mu})$, and $MGE \models \forall.(C \Rightarrow R_j^i = R_k^i)$ if g is $A_j^i = A_k^i$. By Lemma 4.7, $\mathcal{I} \mid \overline{r_j}\langle f\rangle(\lambda\vec{d})\mathbf{0} \rightarrow^{\star} \mathcal{I}' \mid \mathbf{0}\{\vec{d'}/\vec{d}\}$ for some $\vec{d'}$ and $\mathcal{I} \mid (\nu s)(\mathsf{p_unif}[s, r_j, r_k] \mid s()) \rightarrow^{\star} \mathcal{I}'$ ($\mathcal{I}' \subseteq \mathcal{I}$), respectively. Since $[\![g\boxed{\mathrm{R}}]\!] = [\![g\boxed{\mathrm{A}}]\!]\{r/a\}$, by T3, $\mathcal{I} \mid [\![g]\!]\{\vec{r}/\vec{a}\} \rightarrow^{\star} \mathcal{I}'$, where g is either $A_j^i = f(\vec{D}^{\mu})$ or $A_j^i = A_k^i$. □

It follows from Proposition 4.9 that the if parts of U2$^-$and U2$^{\pi}$are equivalent.

Proposition 4.10 *If the if parts of U2$^-$ and U2$^\pi$ hold, the antecedent of U2$^-$ holds iff the antecedent of U2$^\pi$ holds*

Proof The proof is obvious from Proposition 4.9 and the above discussion. □

Theorem 4.11 *U2$^-$ iff U2$^\pi$.*

Proof Let us reduce the configuration $\langle [\![u(\vec{R}^\mu)]\!], \mathcal{I} \rangle$ in U2$^\pi$. Since the conditions for transition obtained from the antecedent and the if part of U2$^\pi$ are satisfied, the transition $\langle [\![u(\vec{R}^\mu)]\!], \mathcal{I} \rangle \rightarrow^\star \langle [\![\vec{b}]\!]\theta, \mathcal{I}' \rangle$ is achieved by T6, T4, T1, T2, and Proposition 4.10. The consequent configuration $\langle [\![\vec{b}]\!]\theta, \mathcal{I}' \rangle$ is well-formed and, by Lemma 4.3 and $[\![\cdot]\!]^{-1}$, corresponds to $\langle \vec{b}, C \cup C_g \rangle$. □

4.6 Active Unification

U3$^-$ is the same as U3. $[\![\cdot]\!]$ maps $s = t$ to $[\![s = t]\!]$ one-to-one.

$$\frac{}{\langle [\![s = t]\!], \mathcal{I} \rangle : \mathcal{N} \rightsquigarrow \langle \mathbf{0}, \mathcal{I} \mid [\![s = t]\!] \rangle : \mathcal{N}} \quad \mathbf{U3}^\pi$$

Here $\mathcal{I}$ is well-formed.

Theorem 4.12 *U3$^-$ iff U3$^\pi$.*

Proof From $[\![\cdot]\!]$ and $[\![\cdot]\!]^{-1}$ in Definition 4.4 and the fact that $\mathcal{I} \mid [\![s = t]\!]$ is well-formed, the proof is straightforward. □

Given an FGHC$^-$ program, the theorems 4.8, 4.11, and 4.12 imply that the transition rules U1$^\pi$, U2$^\pi$, and U3$^\pi$ give an operational meaning the same as Ueda's. Consequently, the π-calculus semantics is equivalent to Ueda's operational semantics.

5 Concluding Remarks

This paper presented the π-calculus semantics of moded FGHC and proved that it is equivalent to Ueda's operational semantics of GHC. This paper also introduced FGHC$^-$, which is the source language for the translation and has almost the same descriptive power that FGHC does. The translation from FGHC$^-$ to polyadic π-calculus (PPC) was straightforward. Predicates and functor structures were all translated into PPC statements in a uniform way. The fact that logical variables in FGHC$^-$ are moded simplified its translation. The moded ground equality theory, rather than Clark's equality theory, was introduced as the theory that the unification in FGHC$^-$ should follow. Then the transition rules for concurrent rewriting, one-step reduction, and active unification in FGHC$^-$ were given. Finally, these transition rules were proved to be equivalent to Ueda's operational semantics.

This semantics should be useful in solving real-world problems, such as efficient execution and language specification.

Acknowledgement: The author thanks Professor Kazunori Ueda for his valuable advice about moded FGHC and Mr. Ken'ichi Yamazaki of NTT for his helpful comments on the earlier draft of this paper. Also the author is grateful to Dr. Rikio K. Onai of NTT for his continuous encouragement.

References

[1] K. Honda and V. Vasconcelos. Guarded Horn Clauses, *unpublished* (1990).

[2] B. Li. A π-calculus Specification of Prolog. In *Proc. of ESOP 1994* (1994).

[3] R. Milner. Functions as Processes. In *Proc. of 17th Int'l Colloquium on Automata, Languages and Programming*, LNCS 443 (1990).

[4] R. Milner. *The Polyadic π-Calculus: a Tutorial.* ECS-LFCS-91-180, University of Edinburgh (1991).

[5] M. Murakami. A Declarative Semantics of Flat Guarded Horn Clauses for Programs with Perpetual Processes. *Theoretical Computer Science*, 75, pp.67–83 (1990).

[6] S.-O. Nyström. Variable-Free Execution of Concurrent Logic Languages. In *Proc. of NACLP'89*, pp.536–552 (1989).

[7] V. Saraswat. Concurrent Constraint Programming, The MIT Press (1993).

[8] K. Ueda. Designing A Concurrent Programming Language. In *Proc. of InfoJapan'90*, Information Processing Society of Japan, pp.87–94 (1990).

[9] K. Ueda and T. Chikayama. Design of the Kernel Language for the Parallel Inference Machine. *The Computer Journal*, Vol.33, No.6, pp.494–500 (1990).

[10] K. Ueda and M. Morita. Moded Flat GHC and Its Message-Oriented Implementation Technique. to appear in *New Generation Computing*, OHMSHA, LTD. (1994).

[11] D. Walker. π-Calculus Semantics of Object-Oriented Programming Languages. In *Proc. of TACS'91*, LNCS 526 (1991).