A Note on a Typing System for the Higher-Order π-Calculus

Vasco T. Vasconcelos
vasco@mt.cs.keio.ac.jp
Department of Computer Science
Keio University
3-14-1 Hiyoshi Kohoku-ku Yokohama 223
Japan

September 22, 1993

Abstract

We study a typing assignment system for the higher-order π-calculus. The system proposed is a straightforward extension of the typing system for the polyadic π-calculus proposed by Kohei Honda and the author [6], by introducing a new type constructor for agents (i.e., processes abstracted on some sequence of names and variables.) We also present an efficient typing reconstruction algorithm to extract the most general typing of an agent or to detect its inexistence, and prove its correctness with respect to the typing system. Finally we show that well-sorted agents are typable, and that typable agents are well-sorted, thus establishing the correspondence between Sangiorgi's sorting system [4] and the typing assignment system.

Introduction

The present report is the summary of a brief investigation on the nature of the name/agent-variable structures that arise in the higher-order π-calculus [4]. The result that types (in the sense of [6]) for higher-order agents do not add much to what was already know for the polyadic π-calculus [6, 7] is in conformity with Sangiorgi's conclusions at the behavior level [4].

To extend the framework of types for the polyadic π-calculus into the higher-order scenario, all we have to do is to introduce a new type constructor $[\alpha_1 \cdots \alpha_n]_n$ describing a process abstracted in a sequence of n names of types $\alpha_1, \ldots \alpha_n$, respectively. Recall that the type of a name capable of carrying a sequence of n names of types $\alpha_1, \ldots \alpha_n$ is denoted by $(\alpha_1 \cdots \alpha_n)_n$. The typing system proposed in [6, 7] adapts easily to the new setting, and so does the typing reconstruction algorithm.

Sangiorgi describes a higher-order π-calculus where legal agents must be syntactically correct (we call these pre-agents) and must verify a system of formation rules. In contrast, we present a system of inference rules, assigning typings to pre-agents. To establish the relation between the two approaches, we prove that *agents are typable* and, conversely, that *typable pre-agents are agents*. This is achieved through a technique, *sort-flattening*, that eliminates the nesting of parenthesis in object-sorts (thus making the sorting function a system of regular equations), and then applying the techniques in [6, 7]. The idea is to introduce a new kind of object-sorts for agents, of the form $[s_1 \cdots s_n]$ representing a process abstracted on names of sorts $s_1, \ldots s_n$.[1]

The outline of this note is as follows. The next section introduces the syntax of the higher-order π-calculus. We show that the higher-order π-calculus without explicit recursion is general enough to express infinite behavior, by means of a fix-point operator. Then, section 2 introduces the notion of types and the typing assignment system. Section 3 presents the notion of principal typings and typing reconstruction. Section 4 studies the relation between types and sorts. The last section concludes the note.

1 The Higher-Order π-Calculus

This section briefly introduces the syntax of the higher-order π-calculus. We also prove the existence of a fix-point operator, thus avoiding the need for explicit constructs for unbounded behavior, in the form of recursion or replication.

Suppose we have an infinite set of *names* and an infinite set of *agent-variables*, or simply *variables*. Let U stand for a name or a variable and K for a name or an agent, and let U^n represent a sequence of n names or variables ($n \geq 0$) and similarly for K^n. The syntax of *processes* $P, Q, \ldots$ and *pre-agents* $A, A', \ldots$ is given by the following grammar.

$$
\begin{array}{lcl}
P & ::= & \mathbf{0} \quad | \quad a(U^n).P \quad | \quad \overline{a}\langle K^n \rangle.P \quad | \quad P + Q \quad | \\
 & & P \mid Q \quad | \quad \nu x P \quad | \quad [a = b]P \quad | \quad X\langle K^n \rangle \\
A & ::= & (U^n)P
\end{array}
$$

Processes of the form $a(U^n).P$ and $\nu x P$, together with agents of the form $(U^n)P$ constitute the binding operators of the calculus. The set of free and bound names is defined accordingly. The operational semantics of the calculus, in the form of a congruence and a reduction relation, $\equiv$ and $\rightarrow$ respectively, can be found in [4]. The relation denoted by $\twoheadrightarrow$ is the reflexive and transitive closure of reduction.

Agents are pre-agents which verify the system of formation rules in [4].

In comparison with the original formulation of the higher-order π-calculus, we notice the absence of constants over processes and equations defining agents, and thus that of explicit recursion. The reason for the omission is that the fragment proposed here is general enough to express a fixed-point operator.

Theorem 1.1 (Fix-Point) *For every process P there is an agent A such that,*

$$A\langle P \rangle \twoheadrightarrow P \mid A\langle P \rangle$$

[1] For the extension of this note, we will write to *sorts* to mean the Milner's [3] or Sangiorgi's [4] approach, and *types* to mean our notion of types [6, 7].

Proof: Take

$$\begin{array}{rcl} PXX & \stackrel{\text{def}}{=} & P \mid \nu q(X\langle q\rangle \mid \overline{q}\langle X\rangle.\mathbf{0}) \\ W & \stackrel{\text{def}}{=} & (p)p(X).PXX \\ WW & \stackrel{\text{def}}{=} & \nu q(W\langle q\rangle \mid \overline{q}\langle W\rangle.\mathbf{0}) \\ A & \stackrel{\text{def}}{=} & (P)\,WW \end{array}$$

Then, A is well-sorted under sorting,

$$\begin{array}{rcl} p, q & : & s \mapsto ((ss)) \\ X & : & (s) \end{array}$$

and,

$$\begin{array}{rcl} A\langle P\rangle & \equiv^2 & \nu q(q(X).PXX \mid \overline{q}\langle W\rangle.\mathbf{0}) \\ & \rightarrow & \nu q(PXX[W/X] \mid \mathbf{0}) \\ & \stackrel{\text{def}}{=} & \nu q((P \mid A\langle P\rangle) \mid \mathbf{0}) \\ & \equiv^3 & P \mid A\langle P\rangle \end{array}$$

□

Notice the similarity with Thomson's paradoxical combinator [5] and with that of the encoding of the λ-calculus paradoxical combinator into the higher-order π-calculus.

2 Type Assignment

This section introduces the notion of types for names and agents, and a typing assignment system assigning types to agents.

2.1 Types and Bases

We have two different kinds of types, namely those for names and those for agents. As in the polyadic π-calculus, the type of a name describes the number and the types of the names the name is able to carry. So, if a name a is capable of carrying a sequence of names of types $\alpha_1, \ldots, \alpha_n$, then the type of a is $(\alpha_1 \cdots \alpha_n)_n$. Similarly, the type of an agent describes the sequence of names on which the agent is abstracted. In this case, if A is an agent abstracted on names and variables $U_1, \ldots U_n$ (that is A is of the form $(U_1 \cdots U_n)P$), and the types of $U_1, \ldots U_n$ are $\alpha_1, \ldots, \alpha_n$, then the type of the agent is $[\alpha_1 \cdots \alpha_n]_n$.

The set of *types* is built from *type variables* $t, t', \ldots$, according to the grammar below, where $n \geq 0$.

$$\alpha \quad ::= \quad t \quad | \quad (\alpha_1 \cdots \alpha_n)_n \quad | \quad [\alpha_1 \cdots \alpha_n]_n$$

We will let $\sigma, \sigma', \ldots$ range over types for names, and $\rho, \rho', \ldots$ range over types for agents. $\alpha, \beta, \ldots$ will range over types in general.

Bases are just sets of name-type and variable-type pairs, where no two names and no two variables occur twice. We will let $B, B', \ldots$ range over bases.

Compatibility of bases plays an important rôle in the type assignment system, allowing to compose bases by a simple set union operation. We say two bases B and B' are *compatible*, written $B \asymp B'$, if and only if,

$$U : \alpha \in B \wedge U : \alpha' \in B' \quad \Rightarrow \quad \alpha = \alpha'$$

2.2 Simple Type assignment

The simple type assignment system to be described, builds from a basis B the type ρ of an agent. *Type assignment statements* are expressions of the form $A \triangleright \rho$. If statement $A \triangleright \rho$ is deducible, from basis B, with the system of rules described below, we say A is *typable*, write $B \vdash A \triangleright \rho$, and call the pair (B, ρ) a *typing* for A.

A little notation helps in simplifying the exposition of the type assignment system. If B is a basis, then $B \cdot U : \alpha$ represents the basis $B \cup \{U : \alpha\}$ provided U does not appear in B. Similarly, $B \cdot U^n : \alpha^n$, represents the basis $B \cdot U_1 : \alpha_1 \cdots U_n : \alpha_n$.

We start with the axiom for names.

$$(\textsc{Name}) \qquad B \cdot x : \sigma \vdash x \triangleright \sigma$$

If A is an agent of of type $[\alpha^n]_n$ and U^m is a sequence of n names and variables of types β^m, then the type of agent $(U^m)A$ is simply $[\alpha^n \beta^m]_{n+m}$. Also, since $(U^m)A$ is a binding operator, we discharge the assumptions on names and variables in U^m.

$$(\textsc{Abs}) \qquad \frac{B \cdot U^m : \beta^m \vdash A \triangleright [\alpha^n]_n}{B \vdash (U^m)A \triangleright [\alpha^n \beta^m]_{n+m}}$$

Applying a sequence K^n of n names and agents of types α^n, to a variable X of type $[\alpha_n \beta_m]_{n+m}$ yields an agent $X\langle K^n \rangle$ of type $[\beta_m]_m$.

$$(\textsc{App}) \qquad \frac{B \vdash K^n \triangleright \alpha_n}{B \vdash X\langle K^n \rangle \triangleright [\beta_m]_m} \qquad (X : [\alpha_n \beta_m]_{n+m} \in B)$$

We now present the rules for plain processes. They look much like the ones in [6], with the exception of rule Emt, where we need to deduce the types of the abstractions (and the names) in the object of the communication.

$$(\textsc{Nil}) \qquad \emptyset \vdash \mathbf{0} \triangleright []_0$$

$$(\textsc{Emt}) \qquad \frac{B \vdash P \triangleright []_0 \qquad B' \vdash K^n \triangleright \alpha^n}{B \cup B' \cup \{a : (\alpha^n)_n\} \vdash \overline{a}\langle K^n \rangle . P \triangleright []_0} \qquad (B \asymp B' \text{ and } B \cup B' \asymp \{a : (\alpha^n)_n\})$$

$$(\textsc{Rcpt}) \qquad \frac{B \cdot U^n : \alpha^n \vdash P \triangleright []_0}{B \cup \{a : (\alpha^n)_n\} \vdash a(U^n) . P \triangleright []_0} \qquad (B \asymp \{a : (\alpha^n)_n\})$$

$$(\textsc{Comp}) \qquad \frac{B \vdash P \triangleright []_0 \qquad B' \vdash Q \triangleright []_0}{B \cup B' \vdash P \mid Q \triangleright []_0} \qquad (B \asymp B')$$

$$(\text{Scop}) \qquad \frac{B \cdot x{:}\sigma \ \vdash P \triangleright [\,]_0}{B \ \vdash \nu x P \triangleright [\,]_0}$$

$$(\text{Match}) \qquad \frac{B \ \vdash P \triangleright [\,]_0}{B \ \vdash [a = b]P \triangleright [\,]_0} \qquad (a{:}\sigma \in B \text{ and } b{:}\sigma \in B)$$

Summation is handled like concurrent composition. Finally, we need a weak rule to introduce in the basis, additional assumptions on names and variables.

$$(\text{Weak}) \qquad \frac{B \ \vdash A \triangleright \rho}{B \cdot U{:}\alpha \ \vdash A \triangleright \rho}$$

There are few important properties the typing system is expected to verify. The first, characteristic of the system proposed in [6], is that,

If $B \ \vdash A \triangleright \rho$, then $B{\restriction}A \ \vdash A \triangleright \rho$.[2]

Subject-reduction is another important property verified by the system, and as a corollary we have that,

If A is typable, then A does not run into type errors.

We say A *runs into a type error* if

$$A \twoheadrightarrow \ldots \mid \overline{a}\langle K^n \rangle.P + \cdots \mid a(U^m).Q + \cdots \mid \ldots$$

and $n \neq m$ or $n = m$ but K_i is a name (resp. an agent) and U_i a variable (resp. a name), for some $1 \leq i \leq n$.

2.3 A Comment on the Form of the Rules

There are many ways to present the rules of a typing system. For example, an alternative rule for emitters could be,

$$\frac{B \ \vdash P \triangleright [\,]_0 \qquad\qquad B \ \vdash K^n \triangleright \alpha^n}{B \ \vdash \overline{a}\langle K^n \rangle.P \triangleright [\,]_0} \qquad (a{:}(\alpha^n)_n \in B)$$

thus avoiding reference to compatibility and union of bases. However, if we want to submit our agents to the typing system as they are (i.e., without having to rename them), and to have deductions of the form $B \ \vdash A \triangleright \rho$ where B contains exactly the free names and variables of A, then it seems we can't avoid compatibility and union of bases. These are exactly two aspects on which our notion of types differs from Milner's sorts, and follow the idea of *building* a typing for an agent, rather than *checking* whether a (renamed) agent is typable against a given basis.

[2] $B{\restriction}A$ represents basis B restricted to the free names and variables in A.

2.4 Recursive Type Assignment

As it happens often with type systems, there are useful and meaningful terms that cannot be typed in the system proposed above. Following [6] we extend the above system to accommodate recursive types of the form,

$$\mu t.\alpha$$

Recursive types are interpreted as infinite trees. A translation from recursive types into infinite trees is described in [6] (cf. [1]). An important property of the thus obtained trees, is that they are *regular*, that is, they have finitely many subtrees.

Let α^* represent the infinite tree associated with type α. We say two types α and β are *equivalent*, written $\alpha \approx \beta$, if the associated trees are equal, that is if $\alpha^* = \beta^*$. The *recursive type assignment system* is obtained by adding to the simple type assignment system, the following rule.

$$(\approx) \qquad \frac{B \cdot U : \alpha \ \vdash A \triangleright \rho}{B \cdot U : \beta \ \vdash A \triangleright \rho} \qquad (\alpha \approx \beta)$$

As an example, the type of the names occurring in the paradoxical combinator is $\mu t.([t]_1)_1$.

3 Principal Typings and Typing Reconstruction

This section presents a notion of principal typings, from which all typings that make an agent typable can be extracted via the notion of instance. An efficient algorithm to extract the principal typing of an agent or to detect the the non-typability of the agent is also presented. The algorithm is a simple adaptation of that in [6], which turn is based on Wand's [8].

3.1 Principal Typings and Instances

The problem of the decidability and computability of typings is formulated through the notions of *principal typing* and *instance*.

We say a typing (B', ρ') is an *instance* of a typing (B, ρ), if there exists a substitution s on types such that $sB \subseteq B'$ and $s\rho = \rho'$. In this case we say (B', ρ') is an instance of (B, ρ) via substitution s. We say a typing (B, ρ) is *principal* for an agent A if and only if,

i. $B \ \vdash A \triangleright \rho$, and

ii. if $B' \ \vdash A \triangleright \rho'$, then the typing (B', ρ') is an instance of (B, ρ).

It is easy to see that the principal typing of an agent, when it exists, is unique up to renaming of type-variables.

3.2 The algorithm

The algorithm to be described accepts an agent A_0 with all bound names renamed to be distinct, and produces a typing (B_0, t_0), and a set of equations E to be submitted to a unification algorithm. Suppose that E is unifiable with a unifier s. Then $s(B_0, t_0)$ is a typing for A_0. On the other hand, if E is not unifiable, then A_0 is not typable. In particular, is s is the most general unifier of E, then $s(B_0 \lceil A_0, t_0)$ is principal for A_0. We will write B_0U for the type associated with name or variable U in basis B_0.

Input: An agent A_0 with all bound names renamed to be distinct.

Initialization: Set $E = \emptyset$, $G = \{(A_0, t_0)\}$ where t_0 is a fresh type variable, and B_0 to a typing assigning to all names and variables in A_0 distinct type-variables.

Loop: If $G = \emptyset$, then halt and return E and (B_0, t_0). Otherwise choose a goal g from G, delete it from G and add to E and G new equations and goals as specified below.

Case (x, t)**:** Generate the equation $t = B_0x$.

Case $((U_1 \cdots U_n)P, t)$**:** Generate the equation $t = [B_0U_1 \cdots B_0U_n]_n$ and the goal P.

Case $(X\langle K_1 \cdots K_n\rangle, t)$**:** Let $t_1, \ldots t_n$ be n fresh type-variables. Generate the equation $t = [t_1 \cdots t_n]_n$ and the goals $(K_1, t_1), \ldots (K_n, t_n)$.

Case 0: Generate nothing.

Case $a(U_1 \cdots U_n).P$**:** Generate the equation $B_0a = (B_0U_1 \cdots B_0U_n)_n$ and the goal P.

Case $\overline{a}\langle K_1 \cdots K_n\rangle.P$**:** Let $t_1, \ldots t_n$ be n fresh type-variables. Generate the equation $B_0a = (t_1 \cdots t_n)_n$ and the goals P and $(K_1, t_1), \ldots (K_n, t_n)$.

Case $P \mid Q$**:** Generate the goals P and Q.

Case νxP**:** Generate the goal P.

Case $[a = b]P$**:** Generate the equation $B_0a = B_0b$ and the goal P.

One interesting result is that the "paradoxical combinator" is not typable in the simple type assignment system. One way to prove this is to run the algorithm on the combinator. It becomes clear that the set of equations produced has no solution with finite types. The interested reader may try to run the algorithm by means of a table, as in [6, 8].

3.3 Correctness of the Algorithm

The correctness of the algorithm is established through an invariant. Some notation helps in the formulation of the invariant.

i. We write $s \models E$ if s unifies E;

ii. Given a basis B_0 we write $s \models (A, t)$ to mean $sB_0 \ \vdash A \triangleright st$ and $s \models P$ to mean $s \models (P, []_0)$, for some agent A and process P;

iii. If G is a set of goals, we write $s \models G$ if $s \models g$, for each goal g in G;

iv. Finally, we say s solves (E, G), and write $s \models (E, G)$, if $s \models E$ and $s \models G$.

Now we can state the invariant of the algorithm.

$$\text{(Soundness)} \qquad \forall s.\ s \models (E, G) \ \Rightarrow\ sB_0 \vdash A_0 \triangleright st_0$$

$$\text{(Completness)} \quad B \vdash A_0 \triangleright \rho \ \Rightarrow\ \exists s.\ s \models (E, G) \wedge (B, \rho) \text{ is an instance of } (B_0 \lceil A_0, t_0) \text{ via } s.$$

The first clause states that any solution for (E, G) generates only correct typings for A_0. The second clause states that every typing for A_0 is generated by some solution for (E, G).

The correctness of the algorithm is established via the below theorems, whose proofs are a simple adaptation of those in [6].

Theorem 3.1 (Termination) *The algorithm always terminates.*

Theorem 3.2 (Correctness) *The invariants are established by the initialization step and preserved by each case in the loop.*

4 Types and Sorts

This section establishes the relationship between the type inference system here proposed and the system of formation rules proposed in [4]. In particular, we prove that,

i. typable pre-agents are agents, and that

ii. agents are typable.

We use a technique to flatten object-sorts, so that the sorting function again represents a *regular system* (cf. [2]), and then we use the techniques in [6, 7], to go from sortings to recursive types and vice-versa.

4.1 Flattening Sorts

When going from the polyadic π-calculus to the HOπ-calculus, Sangiorgi allowed object-sorts to be *non-linear* on the subject-sorts, that is object-sorts could contain arbitrary nested levels of parenthesis. We show that this is not necessary, provided we distinguish object-sorts for names from object-sorts for agents.

As in the case of the polyadic π-calculus, object-sorts for names are simply sequences of subject-sorts enclosed in parenthesis. However, object-sorts for agents become sequences of subject-sorts enclosed in square brackets. In this way the set of object-sorts is defined by the following constructors, where $n \geq 0$.

$$(s_1 \cdots s_n) \quad | \quad [s_1 \cdots s_n]$$

There is a systematic procedure to transform an arbitrary sorting function into a linear sorting function (i.e., a sorting function with images on linear object-sorts.) The idea is to replace non-linear object-sorts of the form $(S_1 \cdots S_n)$ by $(s_1 \cdots s_n)$, where each s_i is,

i. S_i if this is a subject sort, or else,

ii. a fresh subject sort s_i and we extend the sorting function with $s_i \mapsto [s_{i_1} \cdots s_{i_n}]$, if S_i is of the form $(S_{i_1} \cdots S_{i_n})$ and s_{i_j} is the result of applying the procedure to S_{i_j}. Also, whenever $\mathit{Var}(X) = (S_{i_1} \cdots S_{i_n})$ in the original sorting, we make $\mathit{Var}(X) = s_i$ in the linear sorting.

In this way, not only we obtain a linear sorting, but also assignments to names and variables become consistent, that is, we now assign subject-sorts to *both* names and variables.

As an example, the sorting of Plain CHOCS [5] becomes,

$$\begin{array}{rcl} \textsc{Name} & \mapsto & (\textsc{Proc}) \\ \textsc{Proc} & \mapsto & [] \end{array}$$

and that of the names in the "paradoxical combinator",

$$\begin{array}{rcl} s & \mapsto & (s') \\ s' & \mapsto & [s] \end{array}$$

The formation rules for HOπ must now be adjusted to the new linear sorts. This is accomplished by the following procedure.

i. Replace the rule dealing with variables by the following rule,

$$\frac{\mathit{Var}(X) = s \qquad\qquad s \mapsto S}{X : S}$$

ii. Replace, in the formation rule for HOπ, occurrences of $(\tilde{\mathrm{El}})$ and $(\tilde{\mathrm{El}}')$ by $[\tilde{s}']$ and $[\tilde{s}'']$, respectively, *and then* replace occurrences of $\tilde{\mathrm{El}}$ and $\tilde{\mathrm{El}}'$ by $\tilde{s}'$ and $\tilde{s}''$, respectively.

iii. Replace empty object-sorts of the form $()$ by $[]$.

In this way, by performing the replacements above described in any deduction in the original system of formation rules, we obtain a valid deduction in the system of linear sorts.

Lemma 4.1 *If an agent A is well-sorted under a given sorting and the usual formation rules, then A is well-sorted under the corresponding linear sorting and the formation rules for linear sortings.*

4.2 Typable Pre-agents are Agents

This subsection is dedicated to to prove that typable pre-agents are agents. More precisely we prove the following result.

Theorem 4.2 *If A is a pre-agent and there exists a basis B and a type ρ such that,*

$$B \vdash A \triangleright \rho$$

then, there exists a set of subject-sorts, a linear sorting function, a name and a variable assignment to subject-sorts, and a linear object-sort S, such that $A : S$ is valid in the system of formation rules for linear sorts.

To make use of the theory of infinite trees [2], we will map all our types α into the associated infinite tree α^*, but, for simplicity, we will write α instead of α^*.

We first note that if $B \vdash A \triangleright \rho$ and A is renamed to A', so that all bound names and bound variables become distinct, then we have a deduction of $B' \vdash A' \triangleright \rho$ obtained by replacing discharged assumptions on bound names and variables (in rules Rcpt, Scop and Abs) in the original deduction, by using rule Weak. In this way B' contains all variables and names in A'.

Then we identify the set T_s of all subtrees of $T = \{\rho\} \cup \{\alpha \mid U : \alpha \in B'\}$. Notice that, since B' is finite and the trees in T are regular, T_s is finite. Let it be $\{\beta_1, \ldots \beta_k\}$. Notice also that $T \in T_s$.

Now we introduce one unknown s_i for each tree in T_s and define a *system of regular equations*, of the form,

$$\begin{cases} s_1 = u_1 \\ \quad\vdots \\ s_k = u_k \end{cases}$$

where each u_i is

$$\begin{array}{rll} t & \text{if} & \beta_i = t \\ (s_{i_1} \cdots s_{i_l}) & \text{if} & \beta_i = (\beta_{i_1} \cdots \beta_{i_l})_{i_l} \text{ and } s_{i_j} = u_{i_j} \\ [s_{i_1} \cdots s_{i_l}] & \text{if} & \beta_i = [\beta_{i_1} \cdots \beta_{i_l}]_{i_l} \text{ and } s_{i_j} = u_{i_j} \end{array}$$

By [7, 2], this is the unique system of regular equations whose solution contains the trees in T_s.

In this way, let the set of subject-sorts be the set of unknowns introduced above, and let the sorting function map s_i into u_i, for $1 \leq i \leq k$. Assign subject-sort s_j to name x, whenever $x : \beta_j \in B'$, and assign subject-sort s_j to variable X whenever $x : \beta_j \in B'$. Finally, let the object-sort S of agent A be u_j, for $s_j = u_j$ the equation associated with tree ρ.

The result follows by a simple induction on the structure of terms.

4.3 Agents are Typable

This section is dedicated to prove that agents are typable. More precisely, we prove the following result.

Theorem 4.3 *Suppose a linear sorting function is defined over subject-sorts $s_i, \ldots s_k$, and that it maps s_i into S_i, for $1 \leq i \leq k$, and that names x_j are assigned to subject-sorts s_j and variables X_j are assigned to subject-sorts s_j. Further suppose that (renamed) agent A has a sort S under the system of formation rules for linear sorts. Then, there exists a basis B and a type ρ such that,*

$$B \vdash A \triangleright \rho$$

The first step is to solve the system of regular equations $\langle s_1 = S_1, \ldots s_k = S_k \rangle$. Let the sequence of trees $\langle \alpha_1, \ldots \alpha_k \rangle$ be its solution. Such a solution is guaranteed to exist by the results in [2].

Then make $B = \{U_j : \alpha_j \mid U_j \text{ is assigned } s_j\}$. We now prove by induction on the structure of terms that,

$$A : [s_1 \cdots s_n] \quad \Rightarrow \quad B \vdash A \triangleright [\alpha_1 \cdots \alpha_n]_n$$

where α_i is the tree associated with s_i by the above procedure.

Cases when A is of the form $\mathbf{0}$, $P \mid Q$, $P + Q$, $\nu x P$ or $[a = b]P$ are quite simple. We concentrate on the remaining four cases.

Case $(\tilde{U})A$. By the linear formation rules we have $\tilde{U} : \tilde{s}'$ and $A : (\tilde{s})$, and by induction we have $B \vdash A \triangleright [\tilde{\alpha}]$ and $\tilde{U} : \tilde{\beta} \in B$. Then, by rule Abs we have $B/\tilde{U} \vdash (\tilde{U})A \triangleright [\tilde{\alpha}\tilde{\alpha}']$.[3] Finally, we use rule Weak as many times as needed to obtain the result.

Case $X\langle\tilde{K}\rangle$. By the linear formation rules we have $X : s \mapsto [\tilde{s}''\tilde{s}']$ and $\tilde{K} : \tilde{s}'$. By induction, we have $B \vdash \tilde{K} \triangleright \tilde{\alpha}'$ and $X : [\tilde{\alpha}\tilde{\alpha}'] \in B$, where $[\tilde{\alpha}\tilde{\alpha}']$ is the tree associated with s. Then, by rule App we have $B \vdash X\langle\tilde{K}\rangle \triangleright [\tilde{\alpha}]$.

Case $\overline{a}\langle\tilde{K}\rangle.P$. By the linear formation rules we have $\tilde{K} : \tilde{s}$, $P : []$ and $a : s \mapsto (\tilde{s})$. By induction we have $B \vdash \tilde{K} \triangleright \tilde{\alpha}$, and $B \vdash P \triangleright []_0$, and $a : (\tilde{\alpha}) \in B$. Then, by rule Emt we have $B \vdash \overline{a}\langle\tilde{K}\rangle.P \triangleright []_0$, since $B \asymp B$ and $\{a : (\tilde{\alpha})\} \in B$ (and hence $\{a : (\tilde{\alpha})\} \asymp B$).

Case $a(\tilde{U}).P$. By the linear formation rules we have $\tilde{U} : \tilde{s}$, $P : []$ and $a : s \mapsto (\tilde{s})$. By induction, we have $B \vdash \tilde{U} \triangleright \tilde{\alpha}$ and $B \vdash P[]_0$ and $a : (\tilde{\alpha}) \in B$. By rule Rcpt we have $B/\tilde{U} \cup \{a : (\tilde{\alpha})\} \vdash a(\tilde{U}).P[]_0$, since $B/\tilde{U} \asymp \{a : (\tilde{\alpha})\}$, by the fact that $a : (\tilde{\alpha}) \in B$. Finally, we use rule Weak as many times as needed to obtain the result.

Conclusion

We presented a typing assignment system for the higher-order π-calculus, along the lines of the for the polyadic π-calculus. We also presented an efficient algorithm to extract the principal typing of an agent, or to detect its inexistence.

To type agents, we introduced a new type constructor, denoting the number and types of names/variables on which an agent is abstracted. The typing system and the typing inference algorithm of [6] extended smoothly to the higher-order scenario.

To study the relation between sorting and typing, we introduced in the former a new kind of object-sorts, denoting the sorts of the names/variables an agent is abstracted on.

[3] $B/\tilde{U}$ represents basis B with assumptions on names and variables in $\tilde{U}$ removed.

A technique called sort-flattening allows to transform arbitrary object-sorts into linear sorts, thus opening the way to prove, via the theory of infinite trees, that the classes of well-sorted and typable agents coincide.

The fact that the structure of names/variables that arises in the higher-order π-calculus is essentially that of the polyadic π-calculus, reinforces the idea that the ability to transmit processes over channels is not fundamental in process-calculi.

References

[1] Felice Cardone and Mario Coppo. Two Extensions of Curry's Type Inference System. In *Logic and Computer Science*, pages 19–75, Academic Press limited, 1990.

[2] Bruno Courcelle. Fundamental properties of infinite trees. *Theoretical Computer Science*, 25:95–169, 1983.

[3] Robin Milner. *The Polyadic π-Calculus: a Tutorial.* ECS-LFCS 91-180, University of Edinburgh, October 1991.

[4] Davide Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms.* University of Edinburgh, 1992. PhD Thesis.

[5] Bent Thomsen. *Calculi for Higher Order Communicating Systems.* Imperial College of Science, Technolgy and Medecine, 1990. PhD Thesis.

[6] Vasco T. Vasconcelos and Kohei Honda. Principal typing-schemes in a polyadic π-calculus. In E. Best, editor, *Proceedings of CONCUR'93*, pages 524–538, Springer-Verlag, August 1993. LNCS 715.

[7] Vasco T. Vasconcelos and Kohei Honda. *Principal typing-schemes in a polyadic π-calculus.* CS 92-4, Keio University, November 1992.

[8] Mitchell Wand. A simple algorithm and proof for type inference. *Fundamenta Informaticae*, X:115–122, 1987. North-Holland.